

PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

Cz\$ 39,00



INPUT

Vol. 4

Nº 56

NESTE NÚMERO

PROGRAMAÇÃO BASIC

ARMAZENAGEM DE PROGRAMAS

Endereço inicial do BASIC. Como fazer a leitura da memória. Tradução dos códigos. Procurando pelas variáveis. Variáveis numéricas. Cadeias de caracteres e matrizes. Como a memória do computador é utilizada 1101

APLICAÇÕES

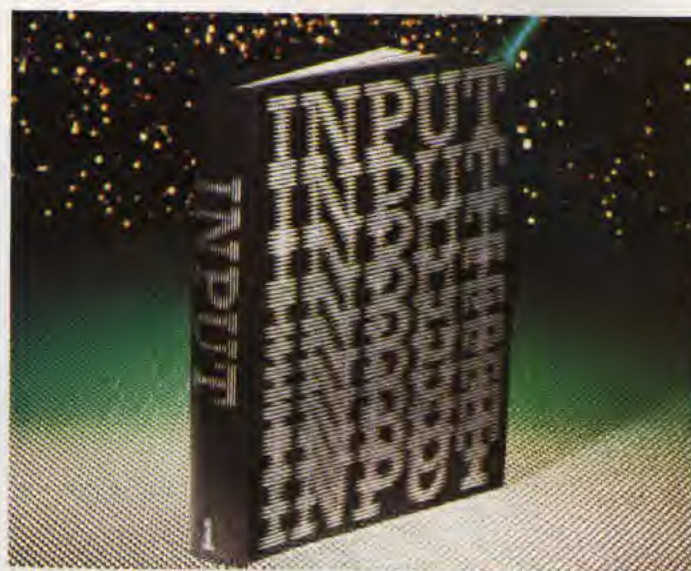
UMA PLANILHA ELETRÔNICA (1)

O que é uma planilha de cálculo? Organize a informação. Por que usar uma planilha. Cálculos feitos pelo computador. Usos da planilha eletrônica. Projeções. Uma planilha típica. Faça sua própria planilha. Planejamento e desenho. Rótulos para as linhas e colunas. Como preencher as células. O começo do programa 1108

CODIGO DE MÁQUINA

AVALANCHE: PEDREGULHOS (1)

Verificação do nível do jogo. Rotina de movimentação da pedra. Animação com duas estruturas. Rolando pela encosta. A pedra chega ao mar. De volta ao topo. Willie foi atingido?..... 1116



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: 1. PESSOALMENTE — Por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em **São Paulo**, os endereços são: rua Brigadeiro Tobias, 773, Centro; avenida Industrial, 117, Santo André; e no **Rio de Janeiro**: avenida Mem de Sá, 191/193, Centro. 2. POR CARTA — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidora Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132, Jardim Teresa — CEP 06000 — Osasco — SP. Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na agência do Correio. 3. POR TELEX — Utilize o nº (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações, Lda. — Qta. Pau Varais, Azinhaga de Fetais — 2 685, Camarate — Lisboa; Apartado 57 — Telex 43 069 JARLIS P.

Atenção: Após seis meses do encerramento da coleção, os pedidos serão atendidos dependendo da disponibilidade do estoque.

Obs.: Quando pedir livros, mencione sempre título e/ou autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.



Editor
VICTOR CIVITA

REDAÇÃO

Diretor Editorial: Carmo Chagas

Editores Executivos: Antonio José Filho,
Berta Sztark Amar

Editor Chefe: Paulo de Almeida

Editora de Texto: Ana Lúcia B. de Lucena

Chefe de Arte: Carlos Luiz Batista

Assistentes de Arte: Dagmar Bastos Sampaio,
Grace Alonso Arruda, Monica Lenardon Corradi

Secretária de Redação/ Coordenadora: Stefania Crema

Secretários de Redação: Beatriz Hagström,

José Benedito de Oliveira Damiano, Maria de Lourdes Carvalho,
Marisa Soares de Andrade, Mauro de Queiroz

COLABORADORES

Consultor Editorial Responsável: Dr. Renato M. E. Sabbatini
(Diretor do Núcleo de Informática Biomédica da
Universidade Estadual de Campinas)

Execução Editorial: DATAQUEST Assessoria em
Informática Ltda., Campinas, SP

Tradução, adaptação, programação e redação:
Abílio Pedro Neto, Aluisio J. Dornellas de Barros,
Marcelo R. Pires Therezo, Marcos Huascar Velasco,
Raul Nader Porrelli, Ricardo J. P. de Aquino Pereira
Coordenação Geral: Rejane Felizatti Sabbatini

COMERCIAL

Diretor Comercial: Roberto Martins Silveira

Gerente Comercial: Flávio Maculan

Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO

Gerente de Produção: João Stungis

Coordenador de Impressão: Afílio Roberto Bonon

Preparador de Texto/Coordenador: Eliel Silveira Cunha

Preparadores de Texto: Alzira Moreira Braz,

Ana Maria Dilguerian, Levon Yacubian,

Luciano Tasca, Maria Teresa Galluzzi,

Maria Teresa Martins Lopes, Paulo Felipe Mendrone

Revisor/Coordenador: José Maria de Assis

Revisoras: Conceição Aparecida Gabriel,

Isabel Leite de Camargo, Lígia Aparecida Ricetto,

Maria de Fátima Cardoso, Nair Lucia de Brito

Paste-up: Anastase Potaris, Balduino F. Leite, Edson Donato

© Marshall Cavendish Limited 1984/85.

© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.

Edição organizada pela Editora Nova Cultural Ltda.

Av. Brigadeiro Faria Lima, nº 2000 - 3º andar

CEP 01452 - São Paulo - SP - Brasil

(Artigo 15 da Lei 5 988, de 14/12/1973).

Esta obra foi composta na AM Produções Gráficas Ltda.
e impressa na Divisão Gráfica da Editora Abril S.A.

ARMAZENAGEM DE PROGRAMAS

■	ENDEREÇO INICIAL
■	TRADUÇÃO DOS CÓDIGOS
■	PROCURANDO PELAS VARIÁVEIS
■	VARIÁVEIS NUMÉRICAS
■	CADEIAS

O funcionamento do computador desperta nossa curiosidade sobretudo quando algo não vai bem. Sabendo como os programas são armazenados, você poderá detectar seus erros com mais facilidade.

O que o computador faz com seu programa depois de tê-lo digitado?

Provavelmente, você sabe que ele é armazenado em um lugar especial da memória, reservado para programas em BASIC. Tem, porém, idéia de onde fica esse lugar e de que maneira as linhas

são escritas, para que o computador as entenda?

Na memória do computador, o programa se assemelha bastante à listagem que você digitou, e muitos dos números ali guardados são códigos ASCII, representando as letras. Mas a máquina re-



duz algumas palavras — as palavras-chave — e, também, introduz informações extras entre as linhas, para facilitar sua localização.

O computador possui ainda uma área onde guarda as variáveis criadas pelo programa, modificando-as sempre que necessário. Quando digitamos, por exemplo, **10 LET A=6**, a máquina não só armazena essa linha como coloca o nome e o valor da variável nela contida em outro local da memória.

Vejam os tudo isso um pouco mais de perto. Para começar, digite o próximo programa exatamente como está. No Spectrum, não deixe nenhum espaço. Nos outros micros, coloque um espaço apenas após as palavras **PRINT** e **LET**.



```
10 PRINT "INPUT"
20 LET A=2
30 PRINT A
40 STOP
```



A armazenagem de um programa em BASIC, no MSX, começa no endereço 32769. Para ler os códigos armazenados, a partir desse endereço, dê o seguinte comando direto:

```
FOR I=32769 TO 32808:PRINT PEEK(I); " ";:NEXT I
```

Obteremos, então, este bloco:

15	128	10	0	145
32	34	73	78	80
85	84	34	0	25
128	20	0	136	32
65	239	19	0	33
128	30	0	145	32
65	0	39	128	40
0	144	0	0	0

Se você considerar esses números como códigos ASCII e tentar relacioná-los aos caracteres equivalentes, descobrirá que só alguns fazem sentido:

?	?	?	?	?
esp	"	I	N	P
U	T	"	?	?
?	?	?	?	esp
A	?	?	?	?
?	?	?	?	esp
A	?	?	?	?
?	?	?	?	?

Já se vê parte do programa. Mas onde estão o **PRINT**, que antecede a palavra **INPUT**, e os demais comandos?

O MSX usa um artifício para economizar memória: codifica as palavras e os caracteres reservados, substituindo-os, ao armazenar o programa, por es-

ses códigos, usualmente chamados *tokens* (do inglês: símbolo, indicação).

Se, posteriormente, você tiver interesse em obter uma lista dos tokens, utilize este programa:

```
10 E=14962
20 C=65
30 PRINT CHR$(C);
40 A=PEEK(E)
50 B=PEEK(E+1)
60 AS=CHR$(A)
70 IF A<128 THEN PRINT AS;:GOTO 110
80 PRINT CHR$(A-128);TAB(8);B
90 E=E+1
100 IF PEEK(E+1)<>0 THEN PRINTC
    CHR$(C);
110 IF PEEK(E+1)<>0 THEN 150
120 C=C+1:IF C=89 THEN 170
125 BS=CHR$(C):PRINT
130 IF BS="J" OR BS="Q" THEN 150
140 PRINT BS;
150 E=E+1
160 IF E<=15649 THEN 40
170 E=15654:PRINT
180 A=PEEK(E);B=PEEK(E+1)
190 IF A>127 THEN PRINT CHR$(A-128);
200 IF A<128 THEN PRINT CHR$(A);
210 PRINT TAB(8);B
220 E=E+2:PRINT
230 IF E<=15673 THEN GOTO 180
```

Interpretando os tokens, nosso bloco ficará assim:

?	?	?	?	PRINT
esp	"	I	N	P
U	T	"	?	?
?	?	?	LET	esp
A	=	?	?	?
?	?	?	PRINT	esp
A	?	?	?	?
?	STOP	?	?	?

Na terceira, 17ª, 27ª e 35ª posições encontra-se a sequência 10, 20, 30 e 40. Esses bytes guardam a parte baixa (L) do número de cada linha e cada byte subsequente guarda a parte alta (H). Os dois bytes anteriores armazenam, do mesmo modo, o endereço da próxima linha.

120L	120H	10	0	PRINT
esp	"	I	N	P
U	T	"	?	130L
130H	20	0	LET	esp
A	=	?	?	140L
140H	30	0	PRINT	esp
A	?	fimL	fimH	40
0	STOP	?	?	?

Para os bytes finais, o valor 0 significa fim de linha; dois 0 seguidos indicam fim de programa. O valor 2, atribuído à variável **A**, é armazenado no 23º byte, de uma forma relativamente

complicada, que não cabe aqui examinar. Finalmente, temos o diagrama completo de uma linha na RAM:

120L	120H	10	0	PRINT
esp	"	I	N	P
U	T	"	fimL	130L
130H	20	0	LET	esp
A	=	2	fimL	140L
140H	30	0	PRINT	esp
A	fimL	fimL	fimH	40
0	STOP	fimL	fimpr	fimpr

VARIÁVEIS NUMÉRICAS

Já sabemos como um programa é colocado na memória do micro. Veremos agora como o valor de uma variável é armazenado na RAM. Lembre-se de que o MSX possui a função **VARPTR** (do inglês **VARiable PoinTeR**), que mostra o endereço a partir do qual está guardada determinada variável.

Vamos explorar, em primeiro lugar, as variáveis numéricas.

Use **NEW** para apagar o programa anterior e execute este:

```
10 DEFINT A
20 A=1
30 V=VARPTR(A)-2
40 FOR K=V TO V+3
50 PRINT PEEK(K);
60 NEXT
```

Você obterá os seguintes números:

```
65 0 1 0
```

O primeiro e o segundo bytes armazenam o nome da variável — por isso, só as duas primeiras letras dela importam. Como se trata de uma variável do tipo inteira, bastam os dois bytes seguintes para guardar seu valor nas duas diferentes modalidades (parte baixa, L, e parte alta, H).

Faça as seguintes substituições no programa e volte a executá-lo:

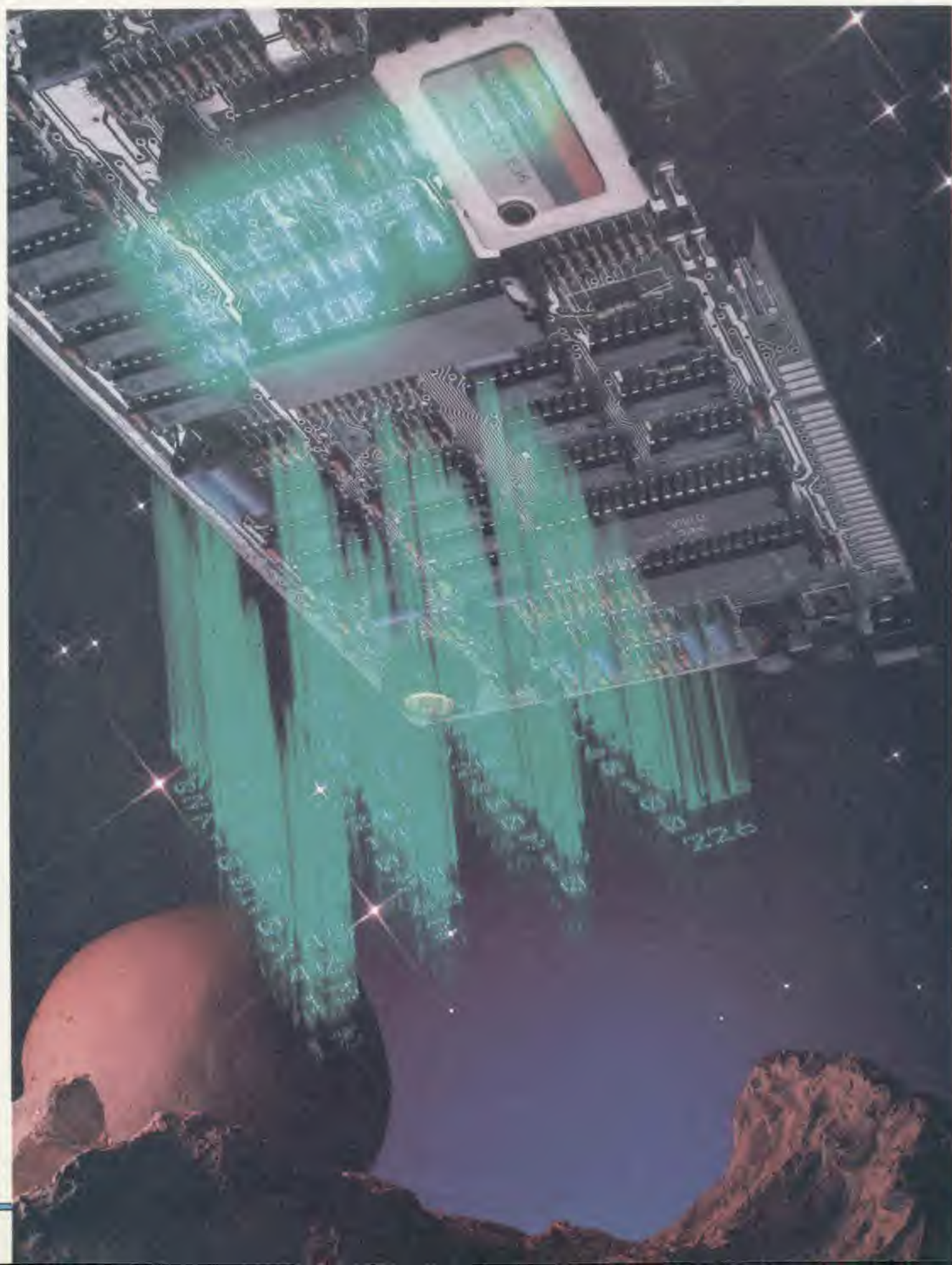
```
10 DEFNSG A
40 FOR K=V TO V+6
```

Temos então estes números:

```
65 0 65 16 0 0
```

Os dois primeiros bytes dão o nome à variável. Estamos lidando agora com uma variável de precisão simples e, por isso, precisaremos de cinco bytes para armazenar seu valor. O primeiro deles cuida do expoente, guardando o resultado da soma de seu valor com o número 65, para facilitar o manejo de expoentes negativos. Os três bytes restantes encarregam-se de guardar o número propriamente dito.

Faça estas novas substituições e execute o programa.




```
10 DEFDBL A
40 FOR K=0 TO 9
```

Aparecem na tela os números:

```
65 0 65 16 0 0 0 0 0 0
```

Como você pode observar, o nome e o expoente continuam sendo armazenados da mesma maneira, mas são necessários sete bytes para guardar o número.

VARIÁVEIS STRING

Para armazenar as variáveis do tipo *string*, o MSX utiliza três bytes. Digite o próximo programa e veja como eles são aproveitados.

```
10 AS="ABC"
20 V=VARPTR(AS)-2
30 FOR K=0 TO 4
40 PRINT PEEK(K); " ";
50 NEXT
```

Você deve ter obtido os números:

```
65 0 3 9 128
```

Os dois primeiros bytes contêm o nome da variável; o terceiro, seu comprimento; os dois últimos indicam o endereço no qual ela se encontra, ou seja, sua posição exata na parte da RAM que armazena o programa. Este é um artifício que o MSX utiliza para economizar memória. Se houver alguma modificação na linha que contém a variável (a linha 10), que impeça o computador de usá-la, ele colocará seu conteúdo em outro endereço.

VARIÁVEIS INDEXADAS

Quando você define uma matriz com o comando **DIM(n)**, o MSX reserva, na memória, um espaço para $n+1$ variáveis, pois a primeira delas tem índice 0. Execute o próximo programa e veja como elas são armazenadas.

```
10 DEFINT A
20 DIM A(2)
30 A(0)=5:A(1)=10:A(2)=15
40 V=VARPTR(A(0))
50 FOR K=0 TO 16
60 PRINT PEEK(K);
70 NEXT K
```

Você deve ter obtido:

```
0 0 0 2 65 0 9 0 1 3
0 5 0 10 0 15 0
```

Os três primeiros bytes zerados inicializam a área de armazenamento. O byte com o número 2 indica que a variável é do tipo inteira — seria 4, para variáveis de precisão simples, e 8, para

variáveis de precisão dupla. Os dois bytes seguintes fornecem o nome da variável. O sétimo e o oitavo indicam quantas posições faltam para o fim dessa área. O nono byte contém o número de dimensões, e o décimo e o 11º, o tamanho dessas dimensões. Em seguida vêm os números, armazenados conforme o tipo de cada variável.

Para as variáveis indexadas do tipo *string* pouca coisa mudaria. O quarto byte, que indica o tipo de variável, teria o número 3, e, em vez de guardar um valor numérico, conteria os endereços dos caracteres, como no armazenamento de strings comuns.

S

No Spectrum, há dois endereços de memória que, juntos, nos dão o endereço a partir do qual está armazenado o programa em BASIC. Acesse-os com o seguinte comando direto:

```
PRINT PEEK 23635+256*PEEK 23636
```

Você obterá como resposta o número 23755, em um Spectrum de 48K. Se você achou um número diferente e tem certeza de que digitou o comando corretamente, substitua 23755 pelo valor encontrado, no próximo comando.

```
FOR I=23755 TO 23755+40:PRINT
PEEK I;" ";NEXT
```

Você obterá os seguintes números:

```
0 10 9 0 245
34 73 78 80 85
84 34 13 0 20
11 0 241 65 61
50 14 0 0 2
0 0 13 0 30
3 0 245 65 13
0 40 2 0 226
13
```

Agora, usando os códigos ASCII, observe como fica o quadro:

```
? ? ? ? ?
" I N P U
T " ? ? ?
? ? ? A =
2 ? ? ? ?
? ? ? ? ?
? ? ? A ?
? ? ? ? ?
?
```

Você já pode ver parte do programa, mas falta decifrar vários códigos.

Provavelmente o comando **PRINT** estaria antes de "INPUT", mas o que vemos nessa posição é o número 245. O Spectrum usa valores acima de 164 para codificar as palavras reservadas. Se

você consultar a lista delas, no manual, verá que 245 é o valor, chamado *token* (do inglês: símbolo, indicação), para **PRINT**. Consultando a lista de tokens, localizaremos os comandos.

```
? ? ? ? PRINT
" I N P U
T " ? ? ?
? ? LET A =
2 ? ? ? ?
? ? ? ? ?
? ? PRINT A ?
? ? ? ? STOP
?
```

No segundo, 15º, 30º e 37º bytes, encontra-se a sequência 10, 20, 30, 40. Esses bytes são usados para guardar a parte baixa (L) do número da linha, assim como o byte anterior guarda a parte alta (H). Os dois bytes seguintes da linha indicam seu tamanho (excluídos os quatro primeiros), sendo que, agora, a parte baixa aparece primeiro.

Temos, assim, o seguinte diagrama:

```
0 10 compl compH PRINT
" I N P U
T " ? 0 20
compl compH LET A =
2 ? ? ? ?
? ? ? 0 30
compl compH PRINT A ?
0 40 compl compH STOP
?
```

Vejam, agora, os bytes finais. O valor 13 é usado para indicar fim de linha (fdl). O número 14 indica que os próximos quatro bytes são uma forma codificada de uma constante numérica — no caso, 2.

```
0 10 compl compH PRINT
" I N P U
T " fdl 0 20
compl compH LET A =
2 cnum 0 0 2
0 0 fdl 0 30
compl compH PRINT A fdl
0 40 compl compH STOP
fdl
```

ARMAZENAGEM DAS VARIÁVEIS

Vimos como um programa em BASIC é armazenado. Todas as variáveis definidas por ele, porém, são guardadas em uma área separada. Dois endereços — 23627 e 23628 — contêm o início dessa área, e dois outros — 23641 e 23642 — apontam para o fim. Para imprimir todo o conteúdo da memória, entre este programa:

```
10 FOR A=PEEK 23627+256*PEEK
23628 TO PEEK 23641+256*PEEK
```



```
23642
20 PRINT PEEK A;" ";
30 NEXT A
```

VARIÁVEIS NUMÉRICAS

Em primeiro lugar, vamos definir uma variável, acrescentando esta linha ao programa anterior.

```
1 LET B=150000
```

Executando o programa, temos:

```
98 146 18 124 0 0 225 0
B  ← valor →
```

O número 98 é o código para a letra **B** mais 32 — isto mostra ao computador que ele está lidando com uma variável numérica. O valor 150000 é armazenado nos cinco bytes seguintes, na forma de ponto flutuante. Os dois últimos bytes, 225 e 0, indicam o fim da área de variáveis. Como eles sempre aparecerão na mesma posição, deixaremos de mencioná-los nos próximos exemplos.

As variáveis com nomes longos são armazenadas da mesma maneira, apesar dos caracteres terem seus códigos um pouco alterados. Tente isto:

```
1 LET MARIA=30
```

Você deve ter obtido os números

```
173 97 114 105 225
M A R I A
0 0 30 0 0
← valor →
```

Os primeiros cinco bytes guardam o nome MARIA. A primeira letra soma-se o número 96, indicando um longo nome; às do meio, o número 32; à última letra é adicionado 160, indicando o fim do nome. Os últimos cinco bytes armazenam o valor numérico.

VARIÁVEIS STRING E INDEXADAS

A próxima linha de programa mostra a maneira como as variáveis do tipo *string* são armazenadas:

```
1 LET AS="STRING"
```

Você deve ter obtido:

```
65 6 0 83 84 82 73 78 71
A S T R I N G
```

O nome da variável, **A**, vem seguido de dois bytes com a quantidade de caracteres armazenados. Como você pode notar, estes caracteres são representados pelos códigos em ASCII.

Tente agora uma variável numérica indexada:

```
1 DIM F(2)
2 LET F(1)=100
3 LET F(2)=200
```

Executando o programa, temos:

```
134 13 0 1 2 0 0 0 100 0 0
F      ← valor →
0 0 200 0 0
← valor →
```

O primeiro byte é **F** mais 64, indicando uma variável numérica indexada. Os dois próximos bytes, com a parte baixa primeiro, indicam o número de bytes seguintes — 13. O próximo byte diz o número de dimensões e os dois seguintes mostram o número de elementos reservados. Só então se chega aos dois valores armazenados, que ocupam cinco bytes cada um.

Veja, finalmente, as variáveis indexadas do tipo *string*. Digite estas linhas e execute o programa.

```
1 DIM BS(2,8)
2 LET BS(1)="NOVA"
3 LET BS(2)="CULTURAL"
```

Observe como fica o quadro:

```
194 21 0 2 2 0 8 0
B
78 79 86 65 32 32 32 32
N O V A
67 85 76 84 85 82 65 76
C U L T U R A L
```

O primeiro byte é o código de **B** mais 128, para variáveis indexadas do tipo *string*. Os dois próximos mostram quantos bytes ainda serão encontrados até o fim da área de variáveis. O byte seguinte guarda o número de dimensões — neste caso, 2. Outros dois pares de bytes contêm, respectivamente, o número de elementos e o comprimento máximo deles. Os bytes restantes guardam os caracteres, sendo que o código 32 (espaço) completa o espaço reservado para um elemento, se este for menor que o comprimento reservado.



No TRS-Color, existem dois endereços que, juntos, indicam o início da área de programas em BASIC. Esses endereços são 25 e 26 (decimal).

Digite o seguinte comando:

```
PRINT PEEK(25)*256+PEEK(26)
```

Se seu computador acabou de ser ligado, a resposta será 7681. Caso obtenha outro número, substitua-o no seguinte comando que irá executar:

```
FOR I=7681 TO 7681+39:PRINT
PEEK(I);:NEXT I
```

Você deve ter obtido este bloco:

```
30 15 0 10 135
32 34 73 78 80
85 84 34 0 30
25 0 20 142 32
65 203 50 0 30
33 0 30 135 32
65 0 30 39 0
40 146 0 0 0
```

Agora, usando os códigos ASCII, observe como fica o quadro:

```
? ? ? ? ?
esp " I N P
U T " ? ?
? ? ? ? esp
A ? 2 ? ?
? ? ? ? esp
A ? ? ? ?
? ? ? ? ?
```

Já podemos ver uma parte do programa em BASIC, mas existem ainda muitos pontos de interrogação.

Você poderia esperar que a palavra **PRINT** aparecesse antes de "INPUT", no diagrama. Em vez disso, temos o número 135. O computador usa valores acima de 127 para representar as palavras reservadas. O número 135 é o valor, também chamado *token* (do inglês: símbolo, indicação), para o comando **PRINT**. Com esses valores, você pode tornar o diagrama mais completo.

```
? ? ? ? PRINT
esp " I N P
U T " ? ?
? ? ? ? LET esp
A = 2 ? ?
? ? ? ? PRINT esp
A ? ? ? ?
? STOP ? ? ?
```

Se quiser, depois, obter a lista dos tokens, use este programa:

```
10 CLEAR 1000:CLS
20 C=43622:FOR K=0 TO 78
30 WS(K/39)=WS(K/39)+CHRS(PEEK(C))
40 C=C+1:IF PEEK(C-1)<128 THEN
30
45 IF K=52 THEN C=33155
50 NEXT
60 C=43802:FOR K=1 TO 34
70 FS=FS+CHRS(PEEK(C))
80 C=C+1:IF PEEK(C-1)<128 THEN
70
85 IF K=20 THEN C=33310
90 NEXT
```



```

100 PRINT:INPUT " DIGITE O SIMBO
LO EM HEXADECIMAL";TS
110 TK=VAL("&H"+TS)
120 IF TK>65280 THEN TK=TK-6528
0:GOTO 210
130 IF TK<128 OR TK>205 THEN 25
0
140 K=-39*(TK>166):P=1
150 IF K=TK-128 THEN WS=WS(K/39
):GOTO 180
160 P=P+1:IF ASC(MIDS(WS(K/39),
P-1,1))<128 THEN 160
170 K=K+1:GOTO 150
180 PRINT:PRINT " SIMBOLO ";TS;
" = ";
190 AS=MIDS(WS,P,1):IF AS<CHRS(
128) THEN PRINT AS:P=P+1:GOTO
190
200 PRINT CHRS(ASC(AS) AND 127)
:GOTO 100
210 K=0:P=1:IF TK<128 OR TK>161
THEN 250
220 IF K=TK-128 THEN WS=FS:GOTO
180
230 P=P+1:IF ASC(MIDS(FS,P-1,1)
)<128 THEN 230
240 K=K+1:GOTO 220
250 PRINT " CARACTER ILEGAL ":PR
INT:GOTO 100

```

Digitando qualquer token, o programa fornecerá a função ou comando equivalente. Seus valores, em hexadecimal, vão de 80 até CD, para os comandos, e de FF80 até FFA1 para as funções.

No quarto, 18°, 28° e 36° bytes há esta sequência: 10, 20, 30, 40. Esses bytes são usados para armazenar a parte baixa (L) do número da linha, enquanto o zero antes deles guarda a parte alta (H).

Os dois primeiros bytes de cada linha são usados para indicar onde começa a linha seguinte. Eles podem ser, por exemplo, 30 e 15. Multiplicando 30 por 256 — pois se trata da parte alta — e adicionando 15, temos 7695, número correspondente ao endereço que contém o primeiro byte da linha 20.

Finalmente, sabendo que o valor 0 significa fim de linha, podemos completar o diagrama:

prx1H	prx1L	0	10	PRINT
esp	"	1	N	P
U	T	"	fd1	prx1H
prx1L	0	20	LET	esp
A	=	2	fd1	prx1H
prx1L	0	30	PRINT	esp
A	fd1	prx1H	prx1L	0
40	STOP	fd1	0	0

ARMAZENAGEM DAS VARIÁVEIS

Vimos como um programa é armazenado. Porém, sempre que se define uma variável no programa, ela é colocada separadamente em duas áreas especiais. Há dois endereços, 27 e 28, que

indicam o início da área utilizada para guardar variáveis simples. Outros dois endereços, 29 e 30, apontam para o início da área de variáveis indexadas.

As variáveis simples também podem ser acessadas por meio do comando **VARPTR**, seguido do nome da variável. Então, **VARPTR(A)** fornecerá a localização do valor da variável **A**. Como o nome dessa variável é armazenado imediatamente antes de seu valor, usaremos **VARPTR(A) - 2**.

VARIÁVEIS SIMPLES

O próximo programa imprime os seis primeiros bytes da área de variáveis simples, mostrando como a variável **A** é armazenada na memória.

```

1 A=1
10 V=VARPTR(A)-2
20 FOR K=V TO V+6
30 PRINT PEEK(K);
40 NEXT:PRINT

```

Você obterá os seguintes números:

```

65 0 129 0 0 0 0
A  ← valor →

```

A letra **A** é armazenada primeiro e o byte seguinte deixa um espaço reservado para outra letra. Os cinco bytes restantes são o número 1, na forma de ponto flutuante.

VARIÁVEIS DO TIPO STRING

As variáveis *string* são armazenadas de forma muito semelhante. Troque a letra **A** da linha 10 por **AA\$** e substitua a linha 1 por:

```
1 AA$="CADEIRA"
```

Desta vez, você terá:

```

65 193 7 0 38 10 0
A A

```

O nome **AA** é colocado nos dois primeiros bytes. Adiciona-se à segunda letra o valor 128, para indicar que ela é uma variável string. O número 7 corresponde à quantidade de bytes usados (conte-os). O 38 e o 10 são, respectivamente, a parte alta e a parte baixa do endereço onde a palavra **CADEIRA** está armazenada.

Calculando esse endereço, temos $38 \times 256 + 10$, que é igual a 9738. Efetivamente, esta é a localização da palavra **CADEIRA** no programa em BASIC. O TRS-Color, desse modo, economiza memória, evitando que a informação seja

duplicada. Mas, assim que se alterar a linha ou a variável, esta será colocada em um outro lugar, e o endereço calculado não será mais o mesmo. Os zeros que cercam os dois números mostram sua utilidade quando o computador apaga velhas variáveis e cria outras.

VARIÁVEIS INDEXADAS

O próximo programa ajudará a vasculhar a área de variáveis indexadas.

```

10 T=1:K=1:V=PEEK(29)*256+PEEK(
30)
20 T=PEEK(V+2)*256+PEEK(V+3)
30 FOR K=V TO V+T-1
40 PRINT PEEK(K);
50 NEXT

```

Acrescente estas linhas e depois execute o programa.

```

1 DIM A(0,2)
2 A(0,0)=10
3 A(0,2)=20

```

Você obterá os seguintes números:

```

65 0 0 24 2 0 3 0 1
A
132 32 0 0 0 0 0 0 0 0
← valor → ← valor →
133 32 0 0 0
← valor →

```

Como sempre, os dois primeiros bytes dão o nome da variável, e os dois outros indicam o seu comprimento. O byte seguinte mostra o número de dimensões, 2, e os dois próximos pares, o número de elementos, na ordem inversa — o 03 e o 01 indicam uma matriz 1 por 3. Depois disso, vêm os próprios valores, na forma de ponto flutuante. Observe que foi reservado um espaço para o elemento (0,1), embora ele não tenha sido definido.

Faça uma experiência com as variáveis string indexadas.

```

1 DIM AS(2)
2 AS(0)="NOVA"
3 AS(2)="CULTURAL"

```

Executando o programa, obterá:

```

65 128 0 22 1 0 3 4 0 38 24 0 0
0 0 0 0 8 0 33 41 0

```

O número 65 corresponde à letra **A** e o 128 indica uma variável string. Há dois bytes para o comprimento, um para a dimensão e um par destinado à quantidade de elementos. Em sequência, encontram-se três grupos de cinco bytes por cadeia. Cada grupo começa por um

número que diz quantos caracteres a variável possui — 4, no caso de **NOVA**. Seguem-se um zero para guardar uma casa, dois bytes para o endereço da variável e mais um zero. O mesmo se repete para todas as variáveis, inclusive as que não foram definidas. Os valores 38 e 33 poderão ser diferentes se se colocar o programa em uma parte diferente da memória.



Tanto o Apple como o TK-2000 começam a armazenar os programas em BASIC a partir do endereço 2048. Depois de digitar o programa inicial, execute o seguinte comando para ver como ele foi colocado na memória:

```
FOR I=2048 TO 2083:PRINT PEEK(I); " ";:NEXT
```

Você obterá estes números:

0	14	8	10	0
186	34	73	78	80
85	84	34	0	23
8	20	0	170	65
208	50	0	30	8
30	0	186	65	0
36	8	40	0	179
0				

Vejamos se algo faz sentido, se usarmos os códigos ASCII:

?	?	?	?	?
?	"	I	N	P
U	T	"	?	?
?	?	?	?	A
?	2	?	?	?
?	?	?	A	?
?	?	?	?	?
?				

Talvez você esperasse que a palavra **PRINT** aparecesse antes da palavra **INPUT**. Porém, a fim de economizar memória, o computador substitui as palavras e os caracteres reservados por códigos, denominados *tokens* (do inglês: símbolo, indicação). Consulte a lista de tokens no manual de seu micro e complete o diagrama.

?	?	?	?	?
PRINT	"	I	N	P
U	T	"	?	?
?	?	?	LET	A
=	2	?	?	?
?	?	PRINT	A	?
?	?	?	?	STOP
?				

Já podemos ver uma parte do programa, mas ainda há muitos pontos de interrogação. Se você der uma olhada no quarto, 17º, 26º e 33º bytes, encontra-

rá uma sequência de números — 10, 20, 30, 40. Esses bytes guardam a parte baixa (L) do número de cada linha, e o byte seguinte guarda a parte alta (H). Os dois bytes anteriores contêm o endereço onde se inicia a próxima linha. O valor zero serve para separar as linhas de programa.

```
inic 120H 120L 10 0
PRINT " I N P
U T " fd1 130H
130L 20 0 LET A
= 2 fd1 140H 140L
30 0 PRINT A fd1
fimH fimL 40 0 STOP
fim
```

ARMAZENAGEM DAS VARIÁVEIS

Já vimos como os programas são armazenados. Porém, sempre que se define uma variável, ela é guardada em uma parte especial da memória. O endereço da área de variáveis simples é dado pelos bytes 105 (L) e 106 (H).

Digite este programa e execute-o para ver como elas são armazenadas.

```
1 LET A=3
10 V=PEEK(105)+256*PEEK(106)
20 FOR I=V TO V+6
30 PRINT PEEK(I);
40 NEXT I
```

Teremos, então:

```
65 0 130 64 0 0 0
```

O número 65 é o código ASCII para a letra A, que corresponde ao nome da variável. O byte seguinte tem a mesma finalidade. O terceiro código refere-se ao expoente, e os quatro restantes, finalmente, guardam o valor da variável. A armazenagem dos números, um tanto complicada, não interessa diretamente aos objetivos deste artigo, e, portanto, não será aqui examinada.

Para uma variável do tipo *string*, faça a seguinte substituição:

```
1 LET ABS="APPLE"
```

Executando o programa, você obterá:

```
65 194 5 11 8 0 0
```

Os dois primeiros bytes dão o nome da variável, mas, agora, o valor 128 é adicionado à segunda letra, para indicar uma variável *string*. O terceiro byte guarda seu comprimento. O par de bytes seguintes contém o endereço a partir do qual ela está armazenada. Para economizar memória, em vez de escrever novamente a palavra **APPLE** em outro lugar da memória, o computador guarda apenas a posição que ela ocupa no programa em BASIC: as variáveis *string*,

definidas por um comando **INPUT**, por exemplo, são guardadas em uma outra posição.

VARIÁVEIS INDEXADAS

As variáveis indexadas são armazenadas em um outro endereço, dado pelos bytes 107 e 108. O computador também guarda aí algumas variáveis especiais, como as usadas nos laços **FOR...NEXT**. Por isso, pulamos sete bytes para chegar à variável A.

Execute o programa:

```
1 DIM A(2)
2 A(0)=5:A(1)=10:A(2)=15
10 V=PEEK(107)+256*PEEK(108)
20 FOR I=V+7 TO V+28
30 PRINT PEEK(I); " ";
40 NEXT I
```

Você deve ter obtido a sequência

```
65 0 22 0 1 0 3 131 32 0 0 0
132 32 0 0 0 132 112 0 0 0
```

Os dois primeiros bytes dão o nome da variável. O próximo par diz, na forma LH, quantos bytes foram necessários para o armazenamento (conte-os). O quinto deles guarda o número de dimensões. A partir deste, na ordem decrescente, seriam escritos os tamanhos de cada dimensão; como nós só temos uma, bastam os valores 0 e 3 (note que agora a parte alta veio em primeiro lugar). Finalmente, aparecem os valores, cada um ocupando cinco bytes, como já comentamos.

Para examinar a armazenagem das variáveis *string* indexadas, substitua 28 por 22 na linha 20 e mude as linhas:

```
1 DIM BS(2)
2 BS(1)="NOVA"
3 BS(2)="CULTURAL"
```

Você deve ter tido esta resposta:

```
66 128 16 0 1 0 3 0 0
0 4 23 8 8 40 8
```

Os dois primeiros bytes guardam o nome, como para as variáveis *string* comuns. O par seguinte mostra o espaço ocupado por essa variável (novamente, conte os bytes). O quinto byte guarda o número de dimensões e os valores 0 e 3 indicam, na forma HL, o comprimento da *n*-ésima dimensão (no nosso caso, a única). A partir daí, cada *string* ocupa três bytes, sendo que o primeiro contém seu comprimento e os dois outros, sua posição no programa em BASIC. Os três zeros referem-se à variável **BS(0)**, que nós não definimos.

UMA PLANILHA ELETRÔNICA (1)

Um problema que aflige a maioria das pessoas é o controle de suas despesas. No artigo da página 134, apresentamos um programa bem prático para a organização do orçamento. Veremos, agora, um método desenvolvido a partir da planilha de cálculo utilizada pelos profissionais em contabilidade.

O programa para a planilha eletrônica é extremamente versátil, tendo um potencial quase ilimitado para a manipulação de informações numéricas. E suas aplicações não se restringem à área financeira. Para começar, examinaremos todas as suas possibilidades. Assim, depois de digitar o programa — que será dado em três partes —, você estará apto a usá-lo eficientemente. Instruções detalhadas serão fornecidas junto com as listagens.

O QUE É UMA PLANILHA DE CÁLCULO?

As planilhas de cálculo eletrônicas valem-se de uma das maiores vantagens do computador — sua capacidade de fazer cálculos muito rapidamente. Em essência, mesmo o maior e mais complexo dos computadores é uma máquina de somar. Os computadores, na verdade, só sabem lidar com números — e quem já se aventurou a trabalhar com código de máquina pode confirmar isto.

Bem utilizada, uma planilha eletrônica torna-se uma ferramenta bastante poderosa. Sua aplicação mais freqüente é na área financeira, mas é possível estendê-la a qualquer finalidade. Ela substitui o papel, o lápis e a calculadora, antes usados pelos contadores para prever os lucros de uma empresa ou por cientistas investigando o crescimento populacional. No plano doméstico, pode ser muito útil no controle das despesas ou na organização dos dados relacionados a um hobby.

A planilha tradicional dos contadores, empregada para anotar os lucros e despesas, é uma folha de papel dividida horizontalmente em linhas e verticalmente em colunas. No cabeçalho, em geral, anotam-se os meses do ano, de maneira que cada coluna corresponde a um mês. Ao lado, colocam-se rótulos como receitas e despesas. Para análises

mais detalhadas incluem-se subtítulos, como vendas, exportações, custos operacionais, custos de matéria-prima etc. Cada linha corresponde a um item específico de receita ou despesa.

O título final da página é, usualmente, *Lucros/Perdas* e os números das colunas mostram quanto se ganhou ou se perdeu a cada mês. Na 13.^a coluna, registra-se o total da receita ou despesa, no ano, por área específica.

O preenchimento das células com os números é sempre uma tarefa trabalhosa, quer se recorra ou não ao auxílio de um computador. Mas os contadores que usam planilhas tradicionais vêem-se obrigados, além disso, a fazer as contas dos totais, o que significa somar todos os valores de receita, todos os valores de despesa e, finalmente, calcular a diferença.

O COMPUTADOR ENTRA EM JOGO

Em muitos aspectos, a planilha de cálculo eletrônica é igual à que foi descrita até o momento, dividindo-se também em células formadas por linhas e colunas. Para se obter células de tamanho aceitável, apenas uma pequena parte da planilha é exibida na tela, que é usada como uma "janela" para mostrar uma área particular.

Como na planilha de papel, pode-se colocar qualquer tipo de dado nas células vazias. Elas só adquirem um significado particular depois de definidas, podendo conter títulos, rótulos, valores etc., segundo a necessidade.

Até aqui, a planilha eletrônica é tão ou mais trabalhosa que a manual. Sua grande vantagem reside, de fato, na manipulação dos dados. Atrás de suas células em branco encontra-se uma outra planilha, que diz ao computador o que deve fazer com as informações de cada célula. Essa segunda planilha pode ser checada e corrigida sempre que for necessário.

Para entender melhor como o sistema funciona, voltemos um pouco ao nosso contador tradicional. Suponhamos que ele queira mostrar na primeira coluna o custo de um item; na segunda, a porcentagem de imposto a ser paga so-

Se você vive às voltas com uma grande quantidade de informações numéricas, peça socorro ao seu micro — ele, sem dúvida, se entende melhor com números e contas do que você.

bre aquele valor e, na terceira, a soma dos valores exibidos nas duas colunas anteriores. O computador pode ser programado a fim de executar essa tarefa em instantes. Colocando a instrução adequada em cada célula da coluna dois, ele multiplicará o número da coluna um por uma taxa fixa. Uma instrução semelhante em cada uma das células da coluna três fará com que ele calcule a soma das colunas um e dois.

PROJEÇÕES

Outro problema do contador que trabalha com uma planilha manual é lidar com as alterações dos dados, em geral freqüentes. Um aumento nos custos de mão-de-obra, por exemplo, pode obrigá-lo a recalcular o total de despesas e a revisar todos os valores de *Lucros/Perdas*. Se ele estiver simplesmente registrando valores, o problema não será tão grave. Mas, se seu trabalho incluir a projeção dados dos para um ano ou mais, ele precisará refazer centenas de cálculos.

Executada manualmente, a tarefa será demorada, cansativa e muito vulnerável a erros. Um computador poderá completá-la em alguns décimos de segundo; depois de termos colocado determinados valores na planilha, a mudança de qualquer um provocará o reajuste automático de todos os outros a ele relacionados. Se, por exemplo, o valor correspondente ao custo da matéria-prima muda, o custo total do produto é reajustado de acordo, bem como o lucro final.

Algumas planilhas são capazes de fazer cálculos bem mais complexos, prestando-se, inclusive, à solução de questões do tipo "E se...?", que aparecem constantemente no mundo dos negócios — e em muitas outras áreas. Embora empregadas principalmente para fins comerciais, elas têm utilidade em qualquer campo, sempre que se lida com muitas variáveis interdependentes.

Graças à sua extraordinária versatilidade, as planilhas eletrônicas são um dos tipos mais procurados de programa aplicativo. Muitas delas são compatíveis com outros programas, possibilitando composições adequadas a aplicações de

■	O QUE É UMA PLANILHA DE CÁLCULO?
■	ORGANIZE A INFORMAÇÃO
■	POR QUE USAR UMA PLANILHA
■	CÁLCULOS COM O COMPUTADOR

■	RÓTULOS PARA AS LINHAS E COLUNAS
■	COMO PREENCHER AS CÉLULAS
■	O COMEÇO DO PROGRAMA

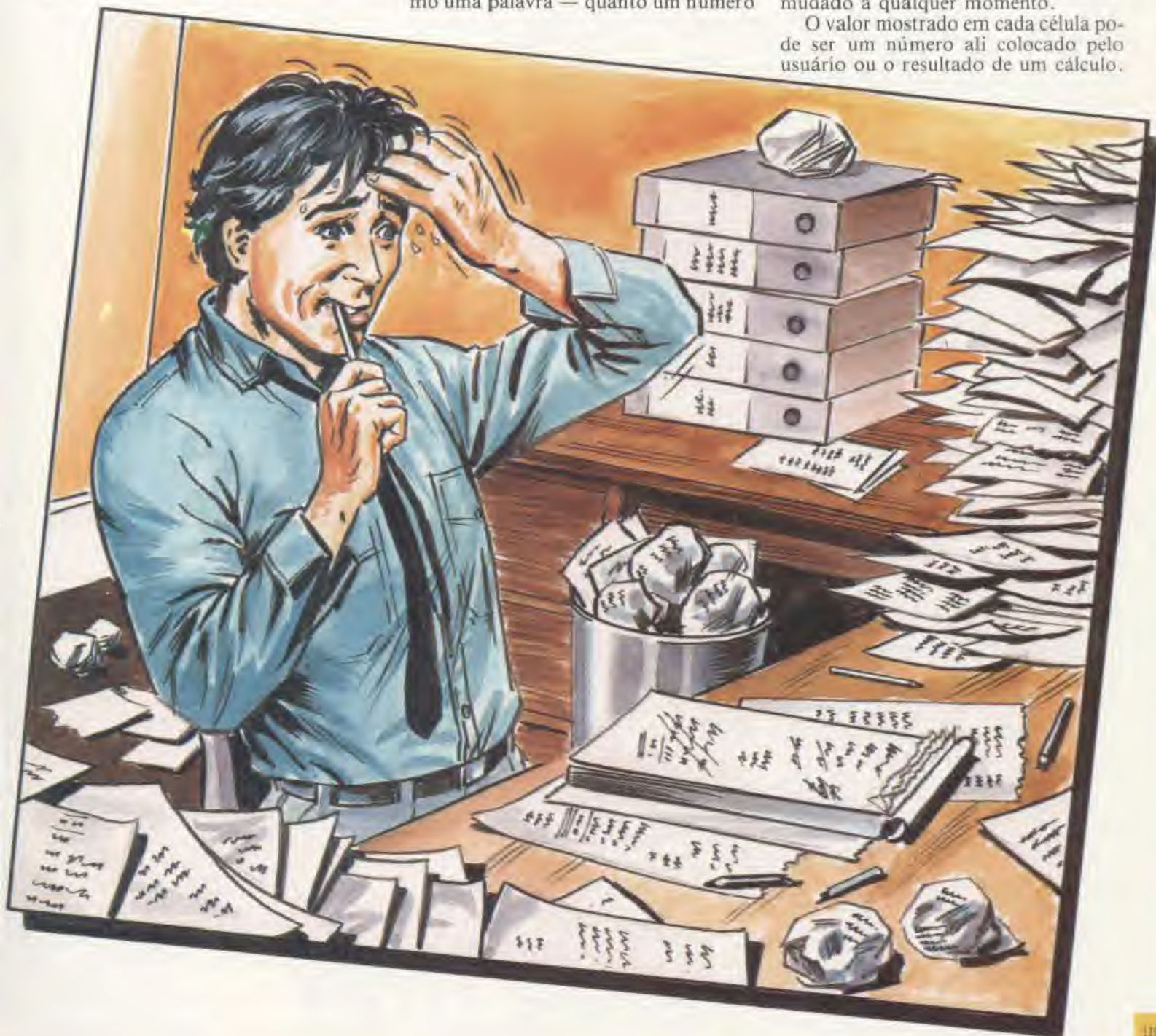
grande complexidade. Um editor de textos, um sistema de gerenciamento de banco de dados e uma planilha eletrônica, por exemplo, constituem um poderoso conjunto de programas.

UMA PLANILHA TÍPICA

A unidade básica da planilha é a célula. O conteúdo de cada célula pode ser tanto uma variável alfanumérica — como uma palavra — quanto um número

ou uma fórmula. Quando a planilha é carregada na memória do computador, as células têm um tamanho predeterminado. Em algumas delas, esse *default* (como é chamado em inglês) pode ser mudado a qualquer momento.

O valor mostrado em cada célula pode ser um número ali colocado pelo usuário ou o resultado de um cálculo.



O número de linhas e colunas varia de uma planilha para outra; as melhores possuem 65 colunas e 256 linhas, ou seja, 16640 células individuais — quantidade excessiva para um pequeno micro manipular! A planilha de INPUT tem 24 colunas e de 20 a 30 linhas, dependendo do computador.

As células são sempre identificadas por letras e números ao longo dos eixos X e Y. A maioria das planilhas usa colunas identificadas por letras: A, B, C, ..., Z e, depois, AA, AB, ... As linhas, nesse caso, são numeradas de 1 em diante. Este é o método usado no nosso programa.

Existem vários comandos disponíveis para a entrada de equações, valores, títulos, assim como para a cópia de células ou o exame de diferentes partes da planilha. Outros comandos fazem os cálculos e permitem que os dados sejam gravados ou carregados para a memória do computador. Podem-se montar equações com as operações elementares, com porcentagens ou com os totais de linhas ou colunas. No micro Spectrum, o movimento de um cursor pela planilha faz com que a célula a ser trabalhada apareça em destaque. Os demais microcomputadores usam um sistema diferente: cada célula é especificada e tem seu conteúdo exibido no rodapé da tela antes de ser transferido para a posição definitiva na planilha.

PLANEJAMENTO E DESENHO

A primeira fase de preparação de uma planilha é a mais difícil, requer detalhado planejamento e não envolve o uso do computador. Antes de mais nada, é necessário estabelecer exatamente o que se quer, pois isso afeta o desenho da planilha. Em seguida, deve-se dedicar o máximo de atenção ao planejamento, para se obter uma planilha que mostre as informações de maneira clara e concisa. Como muitas coisas em computação, esse instrumento será tão bom quanto for seu planejamento. Se você não tiver cuidado nessa fase, provavelmente obterá uma planilha confusa, difícil de ler e até com erros nas fórmulas dos cálculos.

Um exemplo prático: suponhamos que você queira desenhar uma planilha para controlar as finanças domésticas durante o ano. É evidente que os doze meses deverão encabeçar as colunas. Mas decidir o título de cada linha será um pouco mais difícil.

Primeiro: qual o nível de detalhe pretendido? Aluguel, transporte, assistência médica, alimentação são títulos ób-

vios quando se fala de orçamento doméstico. Mas você quer discriminar os gastos com combustível, criando uma categoria independente? Ou prefere reunir todos os gastos relacionados a transporte — além de combustível, manutenção do carro, taxas etc. — em uma categoria única? Tudo depende de seus interesses e necessidades.

Uma planilha pode ser particularmente útil para manter o valor de seus bens atualizado, fornecendo, ainda, informações como as porcentagens de valorização de sua casa ou da depreciação do seu carro.

Calcular a valorização anual de sua casa parece muito simples, inicialmente. Um ano depois de comprá-la, seu valor será o preço de compra multiplicado pela porcentagem de valorização — $P \times X\%$, onde X corresponde à porcentagem mais 100. Por exemplo, X seria $100 + 5\%$ para uma valorização anual de 5%. A fórmula para calcular o valor ao fim do segundo ano seria $P \times X\% \times X\%$. Ano a ano, a fórmula fica maior e mais difícil de ser calculada.

Com uma planilha, o trabalho fica bem mais fácil, e não é preciso ser um matemático, conhecedor de dúzias de fórmulas e equações, para usar todo seu potencial. Num caso como o da valorização da casa, pode-se usar o endereço de uma célula para se referir ao conteúdo dela. Assim, a fórmula nunca fica mais complicada que $P \times X\%$, onde P é o conteúdo da célula anterior. A fórmu-

la a ser escrita na planilha teria esta configuração: **B10%105%**.

Se a fórmula foi posta na célula C10, o resultado é mostrado nesse local. Colocando-se a fórmula **C10*105%** em D10, por exemplo, o computador toma o valor colocado em C10 e o multiplica por 105%.

A apresentação da fórmula varia de uma planilha para outra e os programas deste artigo usam um método bem diferente dos habituais. Os detalhes serão explicados nas instruções sobre uso do programa, dadas mais tarde.

A utilização do endereço da célula em vez de seu conteúdo simplifica o trabalho com a planilha, permitindo que qualquer um, com um pouco de bom senso e paciência, realize tarefas bastante complicadas. No entanto, precisamos ter cuidado ao fazer referência a uma célula em outra. Não devemos, por exemplo, usar a célula B10 em uma fórmula C10, se o resultado de B10 depende do valor assumido por C10. O micro não pode calcular o valor de uma sem ter resolvido a outra! Se a planilha não detectar esse problema, o programa apresentará erro quando o computador tentar resolver o paradoxo.

E SE...?

E se as taxas de juros subirem 15% no mês de junho? E se comprarmos um carro maior? E se instalarmos um aque-



cedor central? Utilizando sua planilha, você poderá obter resposta para todas essas perguntas.

Observe que a última delas aponta para outra área, que não a financeira, na qual as planilhas são úteis. A diferença entre sistemas de aquecimento central com tipos de combustível diversos pode ser visualizada num relance. Estimar a perda de calor que se evita com o uso de dupla isolamento permite também prever quanto vamos economizar e quanto tempo levaremos para recuperar o investimento.

OUTROS USOS

Embora, geralmente, mesmo as planilhas mais simples sejam usadas para aplicações sérias e complicadas, elas se prestam muito bem para o lazer. Vários modelos diferentes dos financeiros po-

dem ser construídos. Só para divertimento, crie uma referência circular com as células que não têm fim.

Existem enormes variações de uma planilha para outra. Como regra geral, quanto mais poderosa, mais cara é a planilha e maior o micro necessário para sua utilização. Uma planilha simples tem uma meia dúzia de comandos e mais ou menos o mesmo número de funções. Compare isso aos vinte comandos e quarenta funções das grandes planilhas. As mais sofisticadas permitem, inclusive, a introdução de declarações com funções similares a comandos em BASIC — por exemplo, IF...THEN, AND, OR e

NOT. Em outras palavras, é possível programar as planilhas.

DIGITE O PROGRAMA

O programa da planilha eletrônica é bastante longo e, por isso, está dividido em três partes. Digite as linhas que se seguem e grave-as para depois adicionar as restantes.

O programa não funciona neste nível. As instruções de como usá-lo serão dadas nas próximas duas partes.

S

```
5 BORDER 0: PAPER 0: INK 7:
CLS
10 DIM bs(11): DIM bs(8): DIM
ds(30,24,18): DIM v(4): DIM
zs(5,4)
20 GOSUB 1730: POKE 23658,8:
LET ts="VAL": LET os=0: LET
bflag=0: LET wx=1: LET wy=1:
LET cx=1: LET cy=1
30 CLS: PRINT "
": FOR x=4 TO
32 STEP 9: FOR y=2 TO 21 STEP
2: PRINT AT y,x;"": NEXT y:
NEXT x: FOR y=1 TO 21 STEP 2:
PRINT AT y,0;"": NEXT y
40 FOR x=0 TO 2: PRINT AT 0,9
*x+0:CHRS (wx+x+64): NEXT x
50 FOR x=0 TO 9: PRINT AT (x+
1)*2,1;(" " AND wy+x<10):wy+x
: NEXT x
60 PRINT AT 0,0:ts;" ": FOR y
=0 TO 9: FOR x=0 TO 2: GOSUB
1230: NEXT x: NEXT y: PRINT
$1:AT 0,0;"
"
70 PRINT AT cy*2,((cx-1)*9)+5
: BRIGHT 1: FLASH 8: PAPER 8:
INK 8: OVER 1;" "
80 IF INKEY$="" AND wy>1
THEN LET wy=wy-10: GOTO 40
90 IF INKEY$="a" AND wy<20
THEN LET wy=wy+10: GOTO 40
100 IF INKEY$="(" AND wx<21
THEN LET wx=wx+3: GOTO 40
110 IF INKEY$=")" AND wx>1
THEN LET wx=wx-3: GOTO 40
120 PRINT AT cy*2,((cx-1)*9)+5
: FLASH 8: BRIGHT 0: INK 8:
PAPER 8: OVER 1;" "
130 LET cy=cy+(INKEY$="6" AND
cy<10)-(INKEY$="7" AND cy>1):
LET cx=cx+(INKEY$="8" AND cx<3
)-(INKEY$="5" AND cx>1)
140 IF INKEY$="i" OR INKEY$="I
" THEN GOSUB 1250
150 IF INKEY$="v" OR INKEY$="V
" THEN LET ts="VAL": GOTO 60
160 IF INKEY$="e" OR INKEY$="E
" THEN LET ts="IGUAL": GOTO 6
0
170 IF INKEY$="?" THEN PRINT
AT 0,0: FLASH 1:"CALC": GOSUB
810: GOTO 60
180 IF INKEY$="z" OR INKEY$="Z
```




```

" THEN PRINT AT 0,0; FLASH 1;
"COPIA": GOSUB 230: GOTO 60
190 IF INKEYS="P" OR INKEYS="p"
" THEN COPY
200 IF INKEYS="NOT " THEN
GOSUB 1780: GOTO 30
210 IF INKEYS="--" THEN GOSUB
1840: GOTO 30
220 GOTO 70
230 PRINT #1;AT 0,0;"CELULA A
COPIAR ?": LET d=1: LET c=3:
LET x=15: GOSUB 580: GOSUB 670:
IF f THEN SOUND .2,30: GOTO
230
240 PRINT #1;AT 0,0;"ABS ou RE
L (A ou R) ?": LET x=22: LET d=
2: LET c=1: GOSUB 580: GOSUB
670: IF f THEN SOUND .2,30:
GOTO 240
250 PRINT #1;AT 0,0;"COL ou LI
NHA (C ou L) ?": LET x=22: LET
d=3: LET c=1: GOSUB 580: GOSUB
670: IF f THEN SOUND .2,30:
GOTO 250
260 PRINT #1;AT 0,0;"DA CELULA
No. ?": LET x=16: LET d=4: LET
c=3: GOSUB 580: GOSUB 670: IF
f THEN SOUND .2,30: GOTO 260
270 PRINT #1;AT 0,0;"PARA A CE
LULA No. ?": LET x=14: LET d=5

```

```

: LET c=3: GOSUB 580: GOSUB
670: IF f THEN SOUND .2,30:
GOTO 270
280 GOSUB 770: IF NOT f THEN
GOTO 320
290 PRINT #1;AT 0,0;"COMANDO E
RRADO - PRESSIONE A PARA AB
ORTAR OU QUALQUER OUTRA TECLA
PARA RE-ENTRAR"
300 PAUSE 10: PAUSE 0: PRINT #
1;AT 0,0;"

```

```

": IF INKEYS="a"
OR INKEYS="A" THEN RETURN
310 GOTO 230
320 LET a=(CODE z$(1,2))-64:
LET b=VAL z$(1,3 TO (VAL z$(1,
1)+1)): LET s$(b,a,9 TO 16
) AND ts="IGUAL")+ (ds(b,a, TO
8) AND ts="VAL"): LET cs=ds(b,
a,17): LET z=CODE ds(b,a,18)
330 IF z$(2,2)="R" THEN GOTO
390
340 FOR a=fc TO tc: FOR b=fr
TO tr
350 IF ts="IGUAL" THEN LET ds
(b,a,9 TO 16)=s$: LET ds(b,a,
17)=cs
360 IF ts="VAL" THEN LET ds(b
,a, TO 8)=s$

```

```

370 NEXT b: NEXT a
380 RETURN
390 LET s$=ds(b,a,9 TO 16):
GOSUB 890: LET a=(CODE z$(4,2)
-64)-(CODE z$(1,2)-64): LET b=
VAL z$(4,3 TO (VAL z$(4,1)+1))
-VAL z$(1,3 TO (VAL z$(1,1)+1)
)
400 LET v(2)=v(2)+b-1: LET v(4
)=v(4)+((b-1) AND v(3)<>26)
410 LET v(3)=v(3)+((a-1) AND v
(3)<>26): LET v(1)=v(1)+a-1
412 IF z$(3,2)="C" THEN LET v
(1)=v(1)+1: LET v(3)=v(3)+(v(3
)<>26)
414 IF z$(3,2)="R" THEN LET v
(2)=v(2)+1: LET v(4)=v(4)+(v(4
)<>26)

```



```

10 PMODE 0,1:PCLEAR 1: CLEAR 100
00:CLS:PRINT @230,"PLANILHA ELE
TRONICA"
20 CS=1:RS=1:CR=1:CC=1:MOS(0)="
VALOR (CALC)":MOS(1)="EQUACAO

```




```

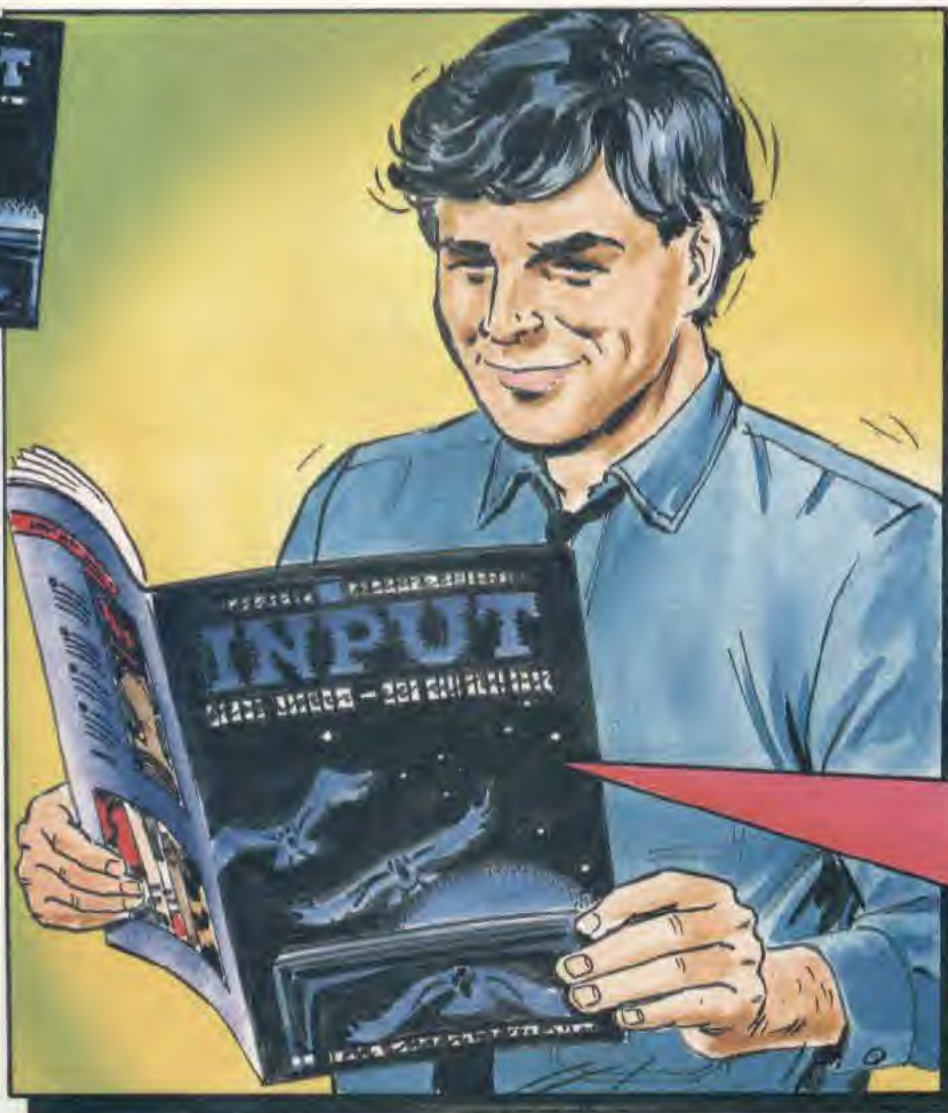
":MO=1:OP$="+-*/%$&"
30 DIM DS(26,30),D(26,30)
40 FOR I=1 TO 26:FOR J=1 TO 30:
DS(I,J)=CHRS(128):NEXT J,I
50 CX=4:RX=1
60 GOSUB 70:GOTO 170
70 PRINT @448,"ESPERE":PRINT @0
,STRINGS(3,128);:FOR I=CS TO CS
+3:PRINT CHRS(123);CHRS(128);CH
RS(128);CHRS(96+I);CHRS(128);CH
RS(128);CHRS(125);:NEXT:PRINT C
HRS(128);
80 PRINT @480,"MODO: ";MOS(MO);
90 FOR I=0 TO 11:C1=INT((RS+I)/
10)+48:C2=(RS+I)-((C1-48)*10)+4
8:POKE 1024+32*I+32,C1:POKE 102
4+32*I+33,C2:PRINT @32*I+34,"":
NEXT
100 PRINT @416:IF MO=0 THEN GOS
UB 740:GOTO 130
110 FOR J=RS TO RS+11:FOR I=CS
TO CS+3
120 PRINT @((J-RS)*32+35+(I-CS)*
7,"":GOSUB 660:NEXT I,J
130 PRINT @480,"MODO: ";MOS(MO)
;TAB(20);"CELULA: ";CHRS(64+CC)
;MIDS(STR$(CR),2);" ";
140 PRINT @448,"PRONTO"
150 PRINT @458,MIDS(DS(CC,CR),2
)
160 RETURN
170 PS=(CR-RS+1)*32+(CC-CS)*7+3
+1024:Z=PEEK(PS):POKE PS,191 AN
D Z
180 IS=INKEY$:IF IS="" THEN 180
190 POKE PS,Z
200 IF IS=CHRS(8) AND CC>1 THEN
CC=CC-1:IF CC<CS THEN CS=CS-1:
GOSUB 70
210 IF IS=CHRS(9) AND CC<26 THE
N CC=CC+1:IF CC>CS+3 THEN CS=CS
+1:GOSUB 70
220 IF IS=CHRS(10) AND CR<30 TH
EN CR=CR+1:IF CR>RS+11 THEN RS=
RS+1:GOSUB 70
230 IF IS=CHRS(94) AND CR>1 THE
N CR=CR-1:IF CR<RS THEN RS=RS-1
:GOSUB 70
240 IF IS="G" THEN GOSUB 330
250 IF IS="Q" THEN CLS:INPUT"CO
NFIRMA QUERER SAIR (S/N) ?":AS:
IF AS<>"S" THEN GOSUB 70 ELSE C

```

```

LS:END
260 IF IS="I" GOSUB 410
270 IF IS="V" THEN MO=0:GOSUB 7
0
280 IF IS="C" GOSUB 1490
290 IF IS="E" THEN MO=1:GOSUB 7
0
300 IF IS="S" GOSUB 1230
310 IF IS="L" GOSUB 1350
320 GOSUB 130:GOTO 170
330 PRINT @448:PRINT @448,"PARA
A CELULA ->";:LINE INPUT AS
340 IF AS="" THEN RETURN
350 C1=ASC(AS)-64:IF C1<1 OR C1
>26 THEN 330
360 C2=VAL(MIDS(AS,2)):IF C2<1
OR C2>30 THEN 330
370 CC=C1:CS=C1:CR=C2:RS=C2
380 IF CS>23 THEN CS=23
390 IF RS>19 THEN RS=19
400 GOSUB 70:RETURN
410 PRINT @448,"NOVO CONTEUDO :
";:LINE INPUT AS
420 IF AS="" THEN AS=CHRS(128):
GOTO 610
430 IF LEN(AS)>9 THEN PRINT @44
8,"ENTRADA INVALIDA":SOUND 1,4:
GOTO 410
440 IF VAL(AS)<>0 THEN 560
450 BS=LEFTS(AS,1):IF BS<"A" OR
BS>"Z" THEN 600
460 CS=MIDS(AS,2,2)
470 IF VAL(CS)<1 OR VAL(CS)>30
THEN 600
480 IF VAL(CS)<10 THEN AS=BS+ST
RS(VAL(CS))+MIDS(AS,3)
490 DS=MIDS(AS,4,1):IF DS<"A" O
R DS>"Z" THEN 600
500 ES=MIDS(AS,5)
510 IF VAL(ES)<1 OR VAL(ES)>30
THEN 600
520 IF VAL(ES)<10 THEN AS=LEFTS
(AS,4)+STR$(VAL(ES))+MIDS(AS,6)
530 OS=MIDS(AS,7,1):IF INSTR(1,
OP$,OS)=0 OR OS="" THEN 600
540 DP=VAL(RIGHTS(AS,1)):IF DP<
0 OR DP>7 THEN 600
550 PRINT @448,"ENTRADA E UMA e
quacao":AS=CHRS(131)+AS:GOTO 61
0
560 PRINT @448,"ENTRADA E UM va
lor"
570 IF RIGHTS(AS,1)="" THEN AS
=LEFTS(AS,LEN(AS)-1):GOTO 570

```




```

580 IF LEN(AS)<7 THEN AS=" "+AS
:GOTO 580
590 AS=CHRS(129)+AS:GOTO 610
600 PRINT @448,"ENTRADA E UMA 1
egenda":AS=CHRS(130)+AS
610 DS(CC,CR)=AS:I=CC:J=CR:PRIN
T @ (J-RS)*32+35+(I-CS)*7,"":GO
SUB 660:SOUND 190.2:FOR D=1 TO
500:NEXT
620 IF CC>CX THEN CX=CC
630 IF CR>RX THEN RX=CR
640 IF MO=0 THEN MO=1:GOSUB 70
650 RETURN

```



```

10 KEYOFF:CLEAR10000:COLOR15,4,
4:SCREEN0:CLS:LOCATE 9,11:PRINT
"PLANILHA ELETRONICA"
20 CS=1:RS=1:CR=1:CC=1:MOS(0)="
VALOR(CALC)":MOS(1)="EQUAÇÃO"
":MO=1:OPS="+-*/%$&
30 DIM DS(26,30),D(26,30)
40 FOR I=1 TO 26:FOR J=1 TO 30:
DS(I,J)=CHRS(128):NEXTJ,I
50 CX=4:RX=1
60 GOSUB 70:GOTO 170
70 CLS:LOCATE 0,20:PRINT"AGUARD
E...":LOCATE0,0:PRINTSTRINGS(3,
219);:FOR I=CS TO CS+4:PRINTCHR
$(91):SPC(2):CHRS(64+I):SPC(2):
CHRS(93):NEXT:PRINTCHRS(219):
80 LOCATE 0,21:PRINT"MOD0: ";MO
$(MO)
90 FOR I=0 TO 15:C1=INT((RS+I)/
10)+48:C2=(RS+I)-((C1-48)*10)+4
8:LOCATE 0,I+1:PRINTCHRS(C1):CH
RS(C2):NEXT
100 LOCATE 19,0:IF MO=0 THEN GO
SUB 740:GOTO 130
110 FOR J=RS TO RS+15:FOR I=CS
TO CS+4
120 LOCATE (I-CS)*7+3,J-RS+1:GO
SUB 660:NEXT I,J
130 LOCATE 0,21:PRINT"MOD0: ";M
OS(MO):TAB(20)"CEL: ";CHRS(64+C
C):MIDS(STRS(CR),2);" "
140 LOCATE 0,20:PRINT"PRONTO!
":TAB(15)MIDS(DS(CC,CR),2):SPC
(9)
160 RETURN
170 PS=(CR-RS+1)*40+(CC-CS)*7+4
:Z=VPEEK(PS):VPOKE PS,219
180 IS=INKEYS:IF IS="" THEN 180
190 VPOKE PS,Z
200 IF IS=CHRS(29) AND CC>1 THE
N CC=CC-1:IF CC<CS THEN CS=CS-1
:GOSUB 70
210 IF IS=CHRS(28) AND CC<26 TH
EN CC=CC+1:IF CC>CS+4 THEN CS=C
S+1:GOSUB 70
220 IF IS=CHRS(31) AND CR<30 TH
EN CR=CR+1:IF CR>RS+15 THEN RS=
RS+1:GOSUB 70
230 IF IS=CHRS(30) AND CR>1 THE
N CR=CR-1:IF CR<RS THEN RS=RS-1
:GOSUB 70
240 IF IS="G" THEN GOSUB 330
250 IF IS="Q" THEN CLS:INPUT"TE
RMINO O PROGRAMA? (S/N) ":AS:IF
AS<>"S" THEN GOSUB 70 ELSECLS:
END

```

```

260 IF IS="I" THEN GOSUB 410
270 IF IS="V" THEN MO=0:GOSUB 7
0
280 IF IS="C" THEN GOSUB 1490
290 IF IS="E" THEN MO=1:GOSUB 7
0
300 IF IS="S" THEN GOSUB 1230
310 IF IS="L" THEN GOSUB 1350
320 GOSUB 130:GOTO 170
330 LOCATE 0,20:PRINTSPC(38):LO
CATE 0,20:PRINT"PARA CEL >":IN
PUT AS
340 IF AS="" THEN RETURN
350 C1=ASC(AS)-64:IF C1<1 OR C1
>26 THEN 330
360 C2=VAL(MIDS(AS,2)):IF C2<1
OR C2>26 THEN 330
370 CC=C1:CS=C1:CR=C2:RS=C2

```

```

380 IF CS>23 THEN CS=23
390 IF RS>19 THEN RS=19
400 GOSUB 70:RETURN
410 LOCATE 0,20:PRINT"NOVO CONT
EUDO: ";SPC(22):LOCATE 15:LINE
INPUT AS
420 IF AS="" THEN AS=CHRS(128):
GOTO 610
430 IF LEN(AS)>9 THEN LOCATE 0,
20:PRINT"ENTRADA INVALIDA
":FOR I=1 TO 500:NEXT:GOTO 410
440 IF VAL(AS)<>0 THEN 560
450 BS=LEFT$(AS,1):IF BS<"A" OR
BS>"Z" THEN 600
460 CS=MIDS(AS,2,2)
470 IF VAL(CS)<1 OR VAL(CS)>30
THEN 600
480 IF VAL(CS)<10 THEN AS=BS+ST
RS(VAL(CS))+MIDS(AS,3)
490 DS=MIDS(AS,4,1):IF DS<"A" O
R DS>"Z" THEN 600
500 ES=MIDS(AS,5)
510 IF VAL(ES)<1 OR VAL(ES)>30
THEN 600
520 IF VAL(ES)<10 THEN AS=LEFTS
(AS,4)+STR$(VAL(ES))+MIDS(AS,6)
530 OS=MIDS(AS,7,1):IF INSTR(1,
OPS,OS)=0 OR OS="" THEN 600
540 DP=VAL(RIGHT$(AS,1)):IF DP<
0 OR DP>7 THEN 600
550 LOCATE 0,20:PRINT"A entrada
é uma EQUAÇÃO":AS=CHRS(131)+AS
:GOTO 610
560 LOCATE 0,20:PRINT"A entrada
é um VALOR"

```




```

570 IF RIGHTS(AS,1)="" THEN AS
=LEFTS(AS,LEN(AS)-1):GOTO 570
580 IF LEN(AS)<7 THEN AS=" "+AS
:GOTO 580
590 AS=CHRS(129)+AS:GOTO 610
600 LOCATE 0,20:PRINT"A entrada
é um ROTULO":AS=CHRS(130)+AS
610 DS(CC,CR)=AS:I=CC:J=CR:LOCA
TE (I-CS)*7+3,(J-RS)+1:GOSUB 66
0
620 IF CC>CX THEN CX=CC
630 IF CR>RX THEN RX=CR
640 IF MO=0 THEN MO=1:GOSUB 70
650 RETURN

```



```

10 TEXT : HOME : CLEAR : VTAB
12: HTAB 10: PRINT "PLANILHA EL
ETRONICA"
20 CS = 1:RS = 1:CR = 1:CC = 1:
CA = 1:RA = 1:MOS(0) = "VALOR(C
ALC)":MOS(1) = "EQUACAO":M
0 = 1:OPS = "+-*/%$&":DS = CHR
$(13) + CHRS(4)
25 ONERR GOTO 2500
30 DIM DS(26,30),D(26,30)
40 FOR I = 1 TO 26: FOR J = 1
TO 30:DS(I,J) = CHRS(128): NE
XT : NEXT
50 CX = 4:RX = 1: FOR I = 1 TO
10:LLS = LLS + CHRS(255): NEX
T
60 GOSUB 70: GOTO 170
70 HOME : VTAB 21: PRINT "AGUA
RDE...": VTAB 1: PRINT LEFTS (
LLS,3): FOR I = CS TO CS + 4:
PRINT CHRS(91): SPC(2): CHRS
(64 + I): SPC(2): CHRS(93):
NEXT : PRINT LEFTS (LLS,2)
80 VTAB 22: PRINT "MODO: ":MOS
(MO):
90 INVERSE : HTAB 1: FOR I = 0
TO 15:C1 = INT ((RS + I) / 10
) + 48:C2 = (RS + I) - ((C1 - 4
8) * 10) + 48: VTAB I + 2: PRIN
T CHRS (C1): CHRS (C2): NEXT :
NORMAL
100 VTAB 19: IF MO = 0 THEN G
OSUB 740: GOTO 130
110 FOR J = RS TO RS + 15: FOR
I = CS TO CS + 4
120 VTAB J - RS + 2: HTAB (I -
CS) * 7 + 4: GOSUB 660: NEXT :
NEXT
130 VTAB 22: HTAB 1: PRINT "MO
DO: ":MOS(MO): TAB(20): "CEL: "
: CHRS(64 + CC): CR: " "
140 VTAB 21: HTAB 1: CALL - 8
68: PRINT "PRONTO!": TAB(15): D
$(CC,CR)
160 RETURN
170 V = CR - RS + 2:H = (CC - C
S) * 7 + 4:VA = RA - RS + 2:HA
= (CA - CS) * 7 + 4
180 VTAB VA: HTAB HA:I = CA:J
= RA: GOSUB 660
182 VTAB V: HTAB H: INVERSE :I
= CC:J = CR: GOSUB 660
183 PRINT : NORMAL
185 CA = CC:RA = CR
190 GET IS

```

	APRIL	MAY	JUNE	JULY	AUGUST	SEPTEMBER	OCTOBER	NOVEMBER	DECEMBER
1	SALES	26170	26170	26170	26170	26170	26170	26170	26170
2	CA. PROFIT	10468	10468	10468	10468	10468	10468	10468	10468
3	OPN	40	40	40	40	40	40	40	40
4	EXPENSE	420	420	420	420	420	420	420	420
5	INVOICE	260	260	260	260	260	260	260	260
6	LOST/ST	470	470	470	470	470	470	470	470
7	RENT/DATE	145	145	145	145	145	145	145	145
8	SUBSIDIES	390	390	390	390	390	390	390	390
9	PHONE	3200	3200	3200	3200	3200	3200	3200	3200
10	WAGES								
11	TOTAL	4930	4930	4930	4930	4930	4930	4930	4930
12	RESULT	5533	5533	5533	5533	5533	5533	5533	5533

Cursor Keys to move : <f4> Large move
<f10> Swap Mode : <f11> Alter cell
<f12> Copy cell : <f13> Calculate
<TAB> to exit : WHOLESCREEN MODE

Como as planilhas tradicionais dos contadores, nossa planilha de cálculo eletrônica também está dividida em células formadas por colunas, que correspondem aos meses do ano, e linhas, equivalentes a rótulos diversos, como, por exemplo, receitas e despesas. Sua grande vantagem reside na forma de manipulação dos dados, que podem ser calculados, checados e alterados com rapidez.

```

200 IF IS = CHRS(8) AND CC >
1 THEN CC = CC - 1: IF CC < CS
THEN CS = CS - 1: GOSUB 70
210 IF IS = CHRS(21) AND CC
< 26 THEN CC = CC + 1: IF CC >
CS + 4 THEN CS = CS + 1: GOSUB
70
220 IF IS = CHRS(90) AND CR
< 30 THEN CR = CR + 1: IF CR >
RS + 15 THEN RS = RS + 1: GOSUB
70
230 IF IS = CHRS(65) AND CR
> 1 THEN CR = CR - 1: IF CR < R
S THEN RS = RS - 1: GOSUB 70
240 IF IS = "G" THEN GOSUB 33
0
250 IF IS = "Q" THEN GOSUB 30
00
260 IF IS = "I" THEN GOSUB 41
0
270 IF IS = "U" THEN MO = 0: G
OSUB 70
280 IF IS = "C" THEN GOSUB 14
90
290 IF IS = "E" THEN MO = 1: G
OSUB 70
300 IF IS = "S" THEN GOSUB 12
30
310 IF IS = "L" THEN GOTO 135
0
320 GOSUB 130: GOTO 170
330 VTAB 21: HTAB 1: CALL - 9
58: PRINT "PARA CEL >": INPUT
AS
340 IF AS = "" THEN RETURN
350 C1 = ASC(AS) - 64: IF C1
< 1 OR C1 > 26 THEN 330
360 C2 = VAL(MIDS(AS,2)): I
F C2 < 1 OR C2 > 30 THEN 330
370 CC = C1:CS = C1:CA = C1:CR
= C2:RS = C2:RA = CR
380 IF CS > 23 THEN CS = 23
390 IF RS > 19 THEN RS = 19
400 GOSUB 70: RETURN
410 VTAB 21: HTAB 1: CALL - 9
58: PRINT "NOVO CONTEUDO: ": I
NPUT AS
420 IF AS = "" THEN AS = CHRS
(128): GOTO 610
430 IF LEN(AS) > 9 THEN VTA

```

```

B 20: PRINT "PALAVRA MUITO GRAN
DE!": CHRS(7): FOR X = 1 TO 30
0: NEXT : GOTO 410
440 IF VAL(AS) < > 0 THEN 5
60
450 BS = LEFTS(AS,1): IF BS <
"A" OR BS > "Z" THEN 600
460 CS = MIDS(AS,2,2)
470 IF VAL(CS) < 1 OR VAL (
CS) > 30 THEN 600
480 IF VAL(CS) < 10 THEN AS
= BS + " " + MIDS(AS,2)
490 DDS = MIDS(AS,4,1): IF DD
$ < "A" OR DDS > "Z" THEN 600
500 ES = MIDS(AS,5)
510 IF VAL(ES) < 1 OR VAL (
ES) > 30 THEN 600
520 IF VAL(ES) < 10 THEN AS
= LEFTS(AS,4) + " " + MIDS (
AS,5)
530 OS = MIDS(AS,7,1): FOR D
= 1 TO LEN(OPS): IF OS < >
MIDS(OPS,D,1) THEN NEXT : GOT
O 600
540 DP = VAL(RIGHTS(AS,1)):
IF DP < 0 OR DP > 7 THEN 600
550 VTAB 21: HTAB 1: CALL - 9
58: PRINT "A entrada e uma EQUA
CAO":AS = CHRS(131) + AS: GOT
O 610
560 VTAB 21: HTAB 1: CALL - 9
58: PRINT "A entrada e um VALOR
"
570 IF RIGHTS(AS,1) = " " TH
EN AS = LEFTS(AS,LEN(AS) -
1): GOTO 570
580 IF LEN(AS) < 7 THEN AS =
" " + AS: GOTO 580
590 AS = CHRS(129) + AS: GOTO
610
600 VTAB 21: HTAB 1: CALL - 9
58: PRINT "A entrada e um ROTUL
O":AS = CHRS(130) + AS
610 DS(CC,CR) = AS: FOR D = 1 T
O 500: NEXT
620 IF CC > CX THEN CX = CC
630 IF CR > RX THEN RX = CR
640 IF MO = 0 THEN MO = 1: GOS
UB 70
650 RETURN

```


AVALANCHE: AS PEDRAS ROLAM (1)

Mais e mais problemas se abatem sobre Willie. Além das serpentes assassinas, dos buracos e da alta da maré, pedras gigantescas rolam morro abaixo, ameaçando soterrá-lo.

Nosso personagem está em grandes dificuldades: enquanto os cabritos monteses devoram seu lanche, ele tenta escalar o penhasco, para não morrer afogado nas águas do mar. Por todo o caminho, buracos e cobras ameaçam sua vida. E, para completar, faremos com que ele se veja diante de uma avalanche de gigantescas pedras!

Teremos bastante trabalho na criação deste novo problema. Precisaremos colocar uma pedra no topo da encosta, empurrá-la morro abaixo, animá-la para que pareça estar rolando, verificar se ela atingiu Willie e, finalmente, apagá-la, quando chegar ao mar. Como esse processo é um tanto complexo, vamos apresentá-lo em duas partes.

S

A primeira parte da rotina movimentará a pedra e dá início à sua descida encosta abaixo. A segunda verifica se Willie foi atingido e coloca a pedra no topo da encosta. Apresentamos aqui apenas a primeira parte. Digite-a, mas não a execute ainda. Incompleta, a rotina não funcionará.

```
10 REM org 58993
20 REM bar ld a, (57344)
30 REM cp 0
40 REM jr z,blm
50 REM cp 3
60 REM jr z,blm
70 REM ret
80 REM blm ld a, (57358)
90 REM cp 1
100 REM jr z,bma
110 REM ld hl, (57356)
120 REM ld bc,57120
130 REM ld a,42
140 REM call 58217
150 REM inc hl
160 REM ld a,45
170 REM ld bc,15616
180 REM call 58217
190 REM ld hl, (57356)
200 REM ld de,480
210 REM sbc hl,de
220 REM jr z,bri
230 REM ld hl, (57356)
240 REM ld de,22560
250 REM add hl,de
260 REM ld a, (hl)
270 REM cp 15
280 REM jr z,bri
290 REM cp 45
```

```
300 REM jr nz,bok
310 REM ld hl, (57356)
320 REM ld bc,15616
330 REM ld a,45
340 REM call 58217
350 REM ld de,32
360 REM add hl,de
370 REM ld (57356),hl
380 REM ld bc,57120
390 REM ld a,42
400 REM call 58217
410 REM bok ld hl, (57356)
420 REM dec hl
430 REM ld (57356),hl
440 REM ld a,1
```

```
450 REM ld (57358),a
460 REM ret
470 REM org 59137
480 REM bri *
490 REM org 59097
500 REM bma *
```

Antes de mais nada, a rotina de movimentação da pedra verifica se a avalanche pode ocorrer no nível atual do jogo. Como você deve se lembrar, Willie tenta não ser alcançado por pedras, saltando sobre elas, no nível um e no nível quatro. A variável correspondente ao ní-



■	VERIFICAÇÃO DO NÍVEL DO JOGO
■	ROTINA DE MOVIMENTAÇÃO DA PEDRA
■	ANIMAÇÃO COM

■	DUAS ESTRUTURAS ROLANDO PELA ENCOSTA
■	A PEDRA CHEGA AO MAR
■	DE VOLTA AO TOPO
■	WILLIE FOI ATINGIDO?

vel atual do jogo é armazenada na posição 57344 da memória; o valor 0 indica o nível um; o valor 3 aponta o nível quatro.

Assim, o conteúdo do endereço 57344 é carregado no acumulador e comparado inicialmente com o valor 0, e, depois, com 3. Se qualquer um desses valores estiver presente, a instrução **jr z,blm** faz com que o processador salte para a rotina de movimentação da pedra. Caso contrário, o processador encontra a instrução **ret** e retorna para a rotina principal do jogo.

QUAL DAS PEDRAS?

Para que possamos animar a pedra, existem na memória dois conjuntos de dados para dois diferentes desenhos da pedra. Quando são impressos alternadamente na tela, eles dão a impressão de que a pedra está rolando.

O processador precisa saber qual foi o último desenho impresso na tela, para poder selecionar adequadamente o próximo. Obtém essa informação no endereço 57358, onde há uma variável cujo conteúdo é 0 ou 1. Esse valor é carregado no acumulador e, se for igual a 1, o processador salta para a rotina **bma**, que será dada mais tarde. Caso o conteúdo do endereço 57358 seja igual a 0, o processador prossegue com a rotina aqui apresentada.

Como você poderá perceber, a variável no endereço 57358 oscilará — se seu conteúdo for 1, será trocado para 0 e vice-versa. Assim, quando o processador voltar a utilizar a rotina de movimentação da pedra, ele irá executar a parte do programa que deixou de lado na vez anterior, imprimindo o outro desenho da pedra.

Naturalmente, o primeiro valor do conteúdo de 57358 é definido pela rotina de inicialização incumbida de ajustar o andamento do jogo.

IMPRIMIR E APAGAR

Os endereços 57356 e 57357 guardam a posição da pedra, cujo valor é carregado no par HL. O par BC é carregado com 57120, que corresponde ao início dos dados do primeiro desenho da pedra. O código 42 (vermelho sobre fundo azul ciano) é carregado em A.

A rotina **print**, que começa no endereço 57217, é chamada. Como de costume, ela imprime os dados apontados pelo conteúdo de BC com a cor especificada pelo conteúdo de A, na posição dada pelo conteúdo de HL.

O par HL é incrementado e passa a apontar para a posição à direita da pedra. Por enquanto, estamos fazendo apenas uma pedra rolar numa parte plana da montanha. Mais tarde, você verá

como utilizar um deslocamento de um espaço para simular a descida da pedra pela encosta.

O acumulador A é carregado com 45 (código da cor ciano sobre fundo ciano) e o par BC, com 15616. Esta posição está na ROM e inicia os dados para um espaço vazio. A rotina **print** é, então, chamada, de novo. A operação faz com que a pedra anteriormente impressa seja apagada. Sem isso, veríamos na tela uma fileira contínua de pedras.

FIM DA LINHA

A próxima tarefa consiste em verificar se a pedra atingiu o canto esquerdo da tela — nessa direção, ela pode ir além da posição 480. Em seguida, o par HL, já incrementado, é recarregado com a posição armazenada em 57356 e 57357. O par DE é carregado com 480 e esse valor é subtraído de HL.

Se o resultado for igual a zero, a pedra está na posição 480 — ou seja, chegou ao fim da descida. A instrução **jr z,bri** manda, então, o processador para a rotina **bri**. Esta, por sua vez, apaga a pedra do final da encosta e reajusta sua posição, mandando-a de volta para o topo da montanha.

Uma vez que não apresentaremos a rotina **bri** no momento, você precisará esperar um pouco mais para ver esta parte do programa funcionar.

CHEGOU AO MAR?

Como a maré está subindo, a pedra poderá chegar à água antes de alcançar o canto esquerdo da tela. Assim, será preciso verificar se a pedra afundou. Caso tenha se chocado contra a água, deverá ser colocada no topo de novo.

O método mais simples para verificar se a pedra atingiu a água consiste em analisar a cor da posição de tela imediatamente abaixo dela. Se for branco sobre azul, a cor do mar, a pedra deve ser apagada. O par de registros HL é novamente carregado com a posição da pedra que se encontra nos endereços 57356 e 57357. O par DE, por sua vez, é carregado com 22560. Na verdade, 22528



posições de memória separam a posição na tela (que está no arquivo de vídeo) da cor que lhe corresponde (que está no arquivo de cores). O valor 32 é adicionado a esse número para que o endereço da cor da posição de tela na linha abaixo — ou 32 caracteres depois — seja localizado.

Os conteúdos de HL e DE são somados em seguida. Sempre se utilizam esses pares de registro para fazer adições ou subtrações entre números de dois bytes, guardando-se o resultado em HL. Portanto, o conteúdo da posição de memória apontada pelo par HL — que agora equivale à posição adequada no arquivo de cores — é carregado no acumulador por meio da instrução de endereçamento indireto **ld a,(hl)**.

Esse valor é, então, comparado a 15, que é o código de branco sobre azul, ou seja, a cor do mar. Se o mar estiver abaixo da pedra, a instrução **jr,zbri** manda o processador para a rotina que apaga a pedra e a coloca de volta no topo da encosta.

RECONHECIMENTO DO DECLIVE

Como já temos no acumulador a cor do caractere abaixo da pedra, vamos aproveitar para verificar se ela está ou não no chão. Para isso, o conteúdo do acumulador é comparado com 45, código correspondente à cor do céu. Se os dois valores forem iguais, esta cor está por baixo da pedra, e o processador continua com a parte seguinte da rotina. Caso contrário, a pedra ainda está firme no chão, e a instrução **jr,nz,bok** faz o processador pular para o rótulo **bok**. Este deixa a pedra onde ela está, e simplesmente acerta as variáveis antes de retornar.

Você deve ter reparado que, até agora, a pedra só tem se movido para a esquerda. Sua posição vertical não foi alterada — ou seja, o declive da encosta não foi levado em conta. Portanto, a pedra só pode estar fora do chão (com a cor do céu sob ela) se tiver passado por uma seção inclinada com um caractere à esquerda — o que significa que precisamos apagá-la e imprimi-la um caractere abaixo. Em linguagem de máquina, tudo isso é feito tão rapidamente que nosso olho — ou a tela de TV — não tem tempo de reagir. Assim, você não poderá ver a pedra em sua posição intermediária, indo para o ar ou voltando para o chão.

Depois que o par de registros HL é carregado com a posição atual da pedra no vídeo, através das posições 57356 e 57357, o par BC é carregado com os da-

dos da ROM para um espaço vazio, como antes. O acumulador A, por sua vez, é carregado com 45 — ciano sobre ciano. A rotina **print** é, então, chamada para apagar a pedra.

O par DE é carregado com 32, valor que se adiciona ao conteúdo HL, o que faz o apontador nesse par de registros mover-se uma posição para baixo na tela. Para ajustar a posição da pedra, o apontador em HL é copiado de volta nos endereços 57356 e 57357. Lembre-se de que a instrução **ld** apenas copia o conteúdo de um endereço ou de um registro em outro endereço ou registro. O conteúdo do lugar de partida permanece inalterado. Logo, quando BC é carregado com o endereço inicial dos padrões da primeira figura da pedra, a nova posição na tela ainda está em HL. O acumulador é carregado com 42 — vermelho sobre ciano — e a rotina **print** é chamada, imprimindo uma pedra vermelha uma posição abaixo da posição anterior de impressão.

ROLANDO PELA ENCOSTA

Quando a pedra é novamente impressa — não importa se uma posição abaixo ou não —, as variáveis têm que ser reajustadas para fazer com que ela pareça estar rolando morro abaixo.

Em seguida, o par HL volta a ser carregado com a posição atual da pedra, pois a rotina **bok** pode ter sido chamada diretamente, não se executando a parte do programa que muda a figura de lugar. Depois de decrementado, o conteúdo de HL é carregado em 57356 e 57357. Na próxima chamada da rotina de movimentação, esse apontador indicará uma posição à esquerda.

Se o conteúdo da variável do tipo de pedra — em 57358 — é 0, o processador continua nesse laço da rotina e imprime a primeira figura. Para que a segunda figura seja impressa, 1 é carregado em 57358 pelo acumulador.

T

A primeira parte da rotina de movimentação da pedra, aqui apresentada, inicia o deslocamento da figura e verifica se ela colide com alguma coisa. Essa rotina não funcionará sem a segunda parte, que continua imprimindo a pedra na tela. Assim, por enquanto, apenas digite e monte estas linhas; não tente executá-las.

```
10  ORG 19789
20  BAR LDA 18238
30  BEQ BLM
```

```
40  CMPA #3
50  BEQ BLM
60  RTS
70  BLM LDX 18253
80  LDU #1536
90  PSHS X
100 JSR CHARPR
110 PULS X
120 LEAX -1,X
130 CMPX #5344
140 BEQ BRI
150 STX 18253
160 LDA ,X
170 CMPA #SAA
180 BEQ BRI
190 CMPA #S55
200 BEQ BNH
210 CMPA #S50
220 BEQ BNH
230 LDA #2
240 STA 18252
250 BNH LEAX 289,X
260 LDA ,X
270 CMPA #SAA
280 BEQ BRI
290 CMPA #S55
300 BNE BOK
310 LEAX -33,X
320 STX 18253
330 CHARPR EQU 19402
340 BOK EQU 19861
350 BRI EQU 19894
```

Antes de mais nada, a rotina de movimentação da pedra verifica se a avalanche pode ocorrer no nível atual do jogo. Como você deve se lembrar, Willie procura não ser atingido pelas pedras, saltando sobre elas, no nível um e no nível quatro. A variável correspondente ao nível do jogo está armazenada na posição de memória 18238; o valor 0 indica o nível um; o valor 3 aponta o nível quatro. Assim, o conteúdo do endereço 18238 é carregado no acumulador; se o valor for 0 ou 3 a instrução **BEQ** faz o processador pular diretamente para a rotina de movimentação da pedra. Caso contrário, o processador encontra a instrução **RET** e retorna para a rotina principal do jogo.

APAGUE A PEDRA

Na posição de memória 18253 está a variável que contém a posição atual da pedra. Essa posição é carregada no registro X, e o número 1536, no registro U. Como U é o apontador da pilha do usuário, a região da memória situada acima do endereço 1536 passa a ser, para todos os fins, a pilha do usuário. A posição 1536 pertence à memória de tela, correspondendo a uma parte do céu. O processador pula, então, para a sub-rotina **CHARPR**.

Lembre-se de que essa sub-rotina uti-

liza os dados da pilha do usuário e imprime na tela um byte de cada vez (um caractere apontado por X). Logo, uma parte do céu é impressa sobre a pedra, apagando-a da tela.

Observe que, antes da rotina **CHARPR** ter sido chamada, o conteúdo de X foi guardado na pilha, mas não removido do registrador — seu valor foi simplesmente copiado. Procedemos assim porque a rotina **CHARPR** pode interferir com o registro X, já que imprime oito bytes de dados que formam um caractere. Ao chamar uma sub-rotina, convém sempre guardar na pilha o conteúdo de um registro que precisamos preservar. A regra é: na dúvida, empilhe, você evitará vários erros.

FIM DA ENCOSTA

Para obter de volta a posição na tela, basta recuperá-la da pilha da máquina. A instrução **LEAX -1,X** decrementa o conteúdo de X, que passa a apontar para uma posição à esquerda. Esse valor é comparado com 5344, endereço da posição do canto esquerdo da tela, onde a pedra irá bater após ter descido toda a encosta.

Se o valor de X for 5344, a pedra atingiu o canto da tela. A instrução **BEQ BRI** manda, então, o processador para a rotina **BRI**, que leva a pedra de volta para o topo da montanha.

Se X não for igual a 5344, a pedra não chegou ao canto da tela. Nesse caso, a instrução **STX 1823** armazena a próxima posição de impressão — que é um caractere à esquerda da anterior na variável de posição da pedra, que está em 1823. É nesta nova posição que a pedra será impressa.

WILLIE FOI ATINGIDO?

Precisamos, também, averiguar se Willie foi atingido pela pedra. Assim, antes que a nova pedra seja impressa, o conteúdo da posição de tela apontada por X é carregado no acumulador por meio da instrução **LDA ,X**. Em seguida, esse valor é comparado com \$55, que é o código de amarelo, a cor do céu, e com \$50, a cor da língua da cobra. Se o caractere contém \$55 ou \$50 — sendo, portanto, parte do céu ou da cobra — o processador pula as duas próximas instruções. Caso contrário, a pedra deve ter atingido Willie — ele é a única figura que pode estar em seu caminho. O salto então não ocorre. O acumulador é carregado com 2. Esse valor é armazenado na variável chamada **dead**, que

indica se Willie morreu ou não. Ela será verificada mais tarde, em outra rotina do jogo.

A PEDRA AFUNDA

A tarefa seguinte consiste em checar se a pedra alcançou a superfície da água. Em caso afirmativo, não será preciso imprimir uma pedra ali: ela terá que ser colocada, novamente, no topo da encosta.

O conteúdo do registro X é adicionado a 289, para indicar a próxima posição na tela, uma linha abaixo. O número 289 resulta da operação $32 \times 8 + 32 + 1$ — ou seja, para chegar à posição imediatamente inferior, devemos contar oito linhas de 32 bytes; como a pedra ocupa uma linha de pixels acima do chão, somamos 32 bytes ao longo da memória de tela; finalmente, adicionamos o número 1 para compensar a subtração anteriormente feita para deslocar o apontador de tela uma posição para a esquerda.

O valor contido nesse caractere é carregado no acumulador por intermédio da instrução **LDA ,X** e, em seguida, comparado com \$AA, que é o código correspondente a azul, a cor do mar. Se o mar está nesse caractere, a instrução **BEQ BRI** salta o processador sobre a rotina que inicializa a volta da pedra à sua posição no topo da encosta.

POSICÃO

Já que é preciso examinar a posição de tela imediatamente inferior, podemos muito bem verificar se existe céu naquela posição. Para isso, basta comparar o seu valor com \$55, o código da cor do céu.

Caso não haja céu onde está a pedra — ou seja, se ela está no chão —, a instrução **BNE BOK** salta para a rotina **BOK**, que imprime a pedra.

Porém, se existe céu por baixo da pedra, ela será movida uma posição para baixo e uma posição para a esquerda. Lembre-se de que agora o apontador de tela está indicando uma linha de pixels abaixo da última posição da pedra. Logo, devemos subtrair o valor 32 para mover uma linha de pixels, e o valor 1, para mover uma posição para a esquerda. Isso é feito de uma só vez pela instrução **LEAX -33,X**.

Para imprimir a pedra no lugar adequado, o processador trabalha diretamente com a rotina **BOK**. Essa rotina, bem como a rotina **BRI**, será exposta num próximo artigo.



A parte da rotina de movimentação da pedra que apresentamos a seguir verifica o nível do jogo e imprime a primeira figura da pedra e, se for o caso, faz com que ela role morro abaixo. Além disso, checa se a pedra chegou ao fim da encosta ou se já alcançou a superfície do mar. A segunda parte da rotina, que daremos num artigo futuro, verifica se Willie foi atingido, imprime a segunda pedra e recoloca a figura no topo da encosta.

Digite e monte a primeira parte, mas não a execute, pois a rotina só funcionará quando estiver completa.

```

10  org 54400
20  ld a, (-5228)
30  cp 0
40  jr z, b1
50  cp 3
60  jr z, b1
70  ret
80  bl ld a, (-5195)
90  cp 1
100 jr z, bm
110 ld hl, (62407)
120 ld de, (-5200)
130 add hl, de
140 ld a, 13
150 push hl
160 call 77
170 pop hl
180 inc hl
190 ld a, 255
200 call 77
210 ld hl, (-5200)
220 ld de, 480
230 sbc hl, de
240 jr z, mo
250 ld hl, (62407)
260 ld de, (-5200)
270 add hl, de
280 ld de, 32
290 add hl, de
300 call 74
310 cp 72
320 jr z, mo
330 cp 76
340 jr z, mo
350 cp 255
360 jr nz, bo
370 ld hl, (62407)
380 ld de, (-5200)
390 add hl, de
400 ld a, 255
410 push de
420 call 77
430 pop de
440 ld hl, 32
450 add hl, de
460 ld (-5200), hl
470 ld de, (62407)
480 add hl, de
490 ld a, 13
500 call 77
510 bo ld hl, (-5200)
520 dec hl

```



```

530 ld (-5200),hl
540 ld a,l
550 ld (-5195),a
560 ret
570 bm call -11006
580 ret
590 mo call -10953
600 ret
610 end

```

Antes de mais nada, a rotina verifica se a avalanche de pedras é necessária no nível atual do jogo. Lembre-se de que Willie tenta escapar das pedras nos níveis um e quatro. A variável que indica o nível do jogo está em -5228 — o valor 0 corresponde ao nível um e o valor 3, ao nível quatro.

O conteúdo desse endereço é carregado no acumulador e comparado inicialmente com 0 e, depois, com 3. Se algum desses valores estiver presente, a instrução **jr z,bl** faz o processador saltar para a rotina de movimentação da pedra. Caso contrário, o processador encontra a instrução **ret** e retorna ao programa principal do jogo.

QUAL DAS PEDRAS?

Para criar o efeito de movimento, utilizamos dois padrões diferentes para a pedra, imprimindo-os alternadamente. O processador precisa saber qual foi a última figura impressa para poder selecionar adequadamente a seguinte. Obtém essa informação no endereço -5195, onde existe uma variável cujo conteúdo é 0 ou 1. Esse valor é carregado no acumulador e, se for igual a 1, o processador salta para a rotina **bm** que será apresentada mais tarde. Caso contrário, ele prossegue com a rotina exposta aqui.

Como você vai perceber, o valor de -5195 é trocado sempre que se chama a rotina de movimentação. Portanto, o processador executa uma parte do programa de cada vez.

IMPRESSÃO

A posição da pedra é armazenada nos endereços -5200 e -5199. Seu valor é carregado no par DE. O endereço inicial da Tabela de Nomes (TN), que está armazenado nos endereços 62407 e 62408, é carregado em HL. A soma dos valores nesse par de registros fornece o valor adequado na TN.

O acumulador A é carregado com o código do padrão que forma o primeiro desenho da pedra. Como você verá na segunda parte da rotina, a outra pe-

dra é formada por dois padrões. Em seguida, a rotina 77 da ROM é chamada, encarregando-se de colocar o valor contido em A no endereço da VRAM apontado pelo par HL — ou seja, a rotina imprime a pedra na tela do computador.

FIM DA ENCOSTA

A etapa seguinte consiste em verificar se a pedra atingiu o fim da encosta. Para isso, sua posição é transferida de -5200 e -5199 para o par HL e o valor 480 é carregado em DE, que representa a última posição que a pedra pode ocupar. Esse valor é subtraído de HL. Se o resultado for zero, a pedra está na posição 480 — em outras palavras, chegou ao fim da encosta. A instrução **jr z,mo** manda, então, o processador para a rotina **mo**, que apresentaremos mais tarde. Essa rotina encarrega-se de apagar a pedra e levá-la de volta para o topo da montanha.

A PEDRA AFUNDOU?

Como a maré está subindo rapidamente, é provável que a pedra atinja a água antes de chegar ao fim do morro.

Precisamos verificar se ela afundou. Em caso afirmativo, a pedra será recolada no topo da encosta.

A rotina 74 da ROM cuida disso. Se a chamarmos com um endereço da VRAM em HL, ela devolverá o conteúdo desse endereço ao acumulador.

Portanto, carregamos o endereço inicial da TN da VRAM em HL. A posição da pedra na tela é carregada em DE. Adicionando os conteúdos de DE e HL, obtemos a posição da pedra na TN. Porém, como estamos interessados no que existe abaixo da pedra, somamos 32 ao conteúdo de HL.

A rotina 74 é chamada e fornece o padrão da figura que se encontra nessa posição. Seu valor é comparado com 72 e 76, que são os códigos dos dois padrões de mar existentes. Se o mar está abaixo da pedra, a instrução **jr z,mo** manda o processador para a segunda parte da rotina, que apaga a pedra e ajusta a posição para o topo do morro.

POSIÇÃO

Uma vez que temos no acumulador o código do padrão que está abaixo da pedra, não custa verificar se ele corresponde ao céu. Para isso, comparamos o conteúdo do acumulador com 255, que é o código do padrão de céu.

Se não há céu abaixo da pedra — ou seja, se ela ainda está no chão — a instrução **jr nz,bo** faz o processador saltar ao rótulo **bo**, onde as variáveis são ajustadas antes de retornar. Caso contrário, o processador continua com a parte seguinte da rotina.

A PEDRA DESCE

Você deve ter reparado que até agora a pedra só tem se movido para a esquerda — o declive da encosta não foi levado em conta. A pedra só pode estar fora do chão (com a cor do céu sob ela) se acabou de passar por uma inclinação — o que significa que precisamos apagá-la e imprimi-la uma posição abaixo. A rapidez com que essa tarefa é executada em código de máquina nos dá a impressão de que a pedra permaneceu na superfície da encosta.

O endereço inicial da TN da VRAM é carregado em HL. O par DE recebe a posição atual da pedra e esse valor é somado em HL. Em seguida o código do padrão de céu é carregado em A e a rotina 77 da ROM é chamada. Com esse procedimento, a pedra é apagada da posição que ocupava na tela.

O valor em DE foi preservado na pilha e somado ao número 32 em HL, para atualizar a posição da pedra nos endereços -5200 e -5199 e levá-la uma posição abaixo da anterior. Finalmente, soma-se o endereço inicial da TN da VRAM em HL e carrega-se o acumulador com o padrão da pedra. A rotina 77 é, então, chamada. Com isso, fizemos a pedra descer uma posição, permanecendo firme junto à encosta.

ROLANDO PELA ENCOSTA

Quer a pedra tenha descido, quer não, as variáveis que definem sua posição devem ser ajustadas para fazer com que ela pareça estar rolando pela encosta da montanha.

O par HL é mais uma vez carregado com a posição da pedra. Em seguida, seu valor é decrementado e devolvido aos endereços -5200 e -5199. Quando a rotina responsável pela movimentação da pedra for chamada novamente, a figura terá sido deslocada uma posição para a esquerda.

O processador continuou a executar essa parte da rotina e imprimiu na tela a primeira figura da pedra porque o conteúdo do endereço -5195 era 0. Para que depois imprima a outra figura, o valor 1 será carregado em -5195 pelo acumulador.

LINHA	FABRICANTE	MODELO	FABRICANTE	MODELO	PAÍS	LINHA
Apple II +	Appietronica	Thor 2010	Appietronica	Thor 2010	Brasil	Apple II +
Apple II +	CCE	MC-4000 Exato	Apply	Apply 300	Brasil	Sinclair ZX-81
Apple II +	CPA	Absolutus	CCE	MC-4000 Exato	Brasil	Apple II +
Apple II +	CPA	Polaris	CPA	Absolutus	Brasil	Apple II +
Apple II +	Digitus	DGT-AP	CPA	Polaris	Brasil	Apple II +
Apple II +	Dismac	D-8100	Codimex	CS-6508	Brasil	TRS-Color
Apple II +	ENIAC	ENIAC II	Digitus	DGT-100	Brasil	TRS-80 Mod.III
Apple II +	Franklin	Franklin	Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Apple II +	Houston	Houston AP	Digitus	DGT-AP	Brasil	Apple II +
Apple II +	Magnex	DM II	Dismac	D-8000	Brasil	TRS-80 Mod. I
Apple II +	Maxitronica	MX-2001	Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Apple II +	Maxitronica	MX-48	Dismac	D-8100	Brasil	Apple II +
Apple II +	Maxitronica	MX-64	Dynacom	MX-1600	Brasil	TRS-Color
Apple II +	Maxitronica	Maxitronic I	ENIAC	ENIAC II	Brasil	Apple II +
Apple II +	Microcraft	Craft II Plus	Engebras	AS-1000	Brasil	Sinclair ZX-81
Apple II +	Milmar	Apple II Plus	Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Apple II +	Milmar	Apple Master	Franklin	Franklin	USA	Apple II +
Apple II +	Milmar	Apple Senior	Gradiente	Expert GPC1	Brasil	MSX
Apple II +	Omega	MC-400	Houston	Houston AP	Brasil	Apple II +
Apple II +	Polymax	Maxxi	Kemitron	Naja 800	Brasil	TRS-80 Mod.III
Apple II +	Polymax	Poly Plus	LNW	LNW-80	USA	TRS-80 Mod. I
Apple II +	Spectrum	Microengenho I	LZ	Color 64	Brasil	TRS-Color
Apple II +	Spectrum	Spectrum ed	Magnex	DM II	Brasil	Apple II +
Apple II +	Suporte	Venus II	Maxitronica	MX-2001	Brasil	Apple II +
Apple II +	Sycomig	SIC I	Maxitronica	MX-48	Brasil	Apple II +
Apple II +	Unitron	AP II	Maxitronica	MX-64	Brasil	Apple II +
Apple II +	Victor do Brasil	Elppa II Plus	Maxitronica	Maxitronic I	Brasil	Apple II +
Apple II +	Victor do Brasil	Elppa Jr.	Microcraft	Craft II Plus	Brasil	Apple II +
Apple IIe	Microcraft	Craft IIe	Microcraft	Craft IIe	Brasil	Apple IIe
Apple IIe	Microdigital	TK-3000 IIe	Microdigital	TK-3000 IIe	Brasil	Apple IIe
Apple IIe	Spectrum	Microengenho II	Microdigital	TK-82C	Brasil	Sinclair ZX-81
MSX	Gradiente	Expert GPC-1	Microdigital	TK-83	Brasil	Sinclair ZX-81
MSX	Sharp	Hotbit HB-8000	Microdigital	TK-85	Brasil	Sinclair ZX-81
Sinclair Spectrum	Microdigital	TK-90X	Microdigital	TK-90X	Brasil	Sinclair Spectrum
Sinclair Spectrum	Timex	Timex 2000	Microdigital	TKS-800	Brasil	TRS-Color
Sinclair ZX-81	Apply	Apply 300	Milmar	Apple II Plus	Brasil	Apple II +
Sinclair ZX-81	Engebras	AS-1000	Milmar	Apple Master	Brasil	Apple II +
Sinclair ZX-81	Filcres	NEZ-8000	Milmar	Apple Senior	Brasil	Apple II +
Sinclair ZX-81	Microdigital	TK-82C	Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Sinclair ZX-81	Microdigital	TK-83	Omega	MC-400	Brasil	Apple II +
Sinclair ZX-81	Microdigital	TK-85	Polymax	Maxxi	Brasil	Apple II +
Sinclair ZX-81	Prologica	CP-200	Polymax	Poly Plus	Brasil	Apple II +
Sinclair ZX-81	Ritas	Ringo R-470	Prologica	CP-200	Brasil	Sinclair ZX-81
Sinclair ZX-81	Timex	Timex 1000	Prologica	CP-300	Brasil	TRS-80 Mod.III
Sinclair ZX-81	Timex	Timex 1500	Prologica	CP-400	Brasil	TRS-Color
TRS-80 Mod. I	Dismac	D-8000	Prologica	CP-500	Brasil	TRS-80 Mod.III
TRS-80 Mod. I	Dismac	D-8001/2	Ritas	Ringo R-470	Brasil	Sinclair ZX-81
TRS-80 Mod. I	LNW	LNW-80	Sharp	Hotbit HB-8000	Brasil	MSX
TRS-80 Mod. I	Video Genie	Video Genie I	Spectrum	Microengenho I	Brasil	Apple II +
TRS-80 Mod.III	Digitus	DGT-100	Spectrum	Microengenho II	Brasil	Apple IIe
TRS-80 Mod.III	Digitus	DGT-1000	Spectrum	Spectrum ed	Brasil	Apple II +
TRS-80 Mod.III	Kemitron	Naja 800	Suporte	Venus II	Brasil	Apple II +
TRS-80 Mod.III	Prologica	CP-300	Sycomig	SIC I	Brasil	Apple II +
TRS-80 Mod.III	Prologica	CP-500	Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
TRS-80 Mod.III	Sysdata	Sysdata II	Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
TRS-80 Mod.III	Sysdata	Sysdata Jr.	Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
TRS-80 Mod.IV	Multix	MX-Compacto	Timex	Timex 1000	USA	Sinclair ZX-81
TRS-80 Mod.IV	Sysdata	Sysdata IV	Timex	Timex 1500	USA	Sinclair ZX-81
TRS-Color	Codimex	CS-6508	Timex	Timex 2000	USA	Sinclair Spectrum
TRS-Color	Dynacom	MX-1600	Unitron	AP II	Brasil	Apple II +
TRS-Color	LZ	Color 64	Victor do Brasil	Elppa II Plus	Brasil	Apple II +
TRS-Color	Microdigital	TKS-800	Victor do Brasil	Elppa Jr.	Brasil	Apple II +
TRS-Color	Prologica	CP-400	Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado. Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:



Sinclair ZX-81



TRS-80



TK-2000



MSX



Spectrum



TRS-Color



Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

PROGRAMAÇÃO BASIC

Simulação e previsão. Simulação de números randômicos.
Amostragem e pesquisa. Simulação de uma loteria esportiva.

PERIFÉRICOS

O que é um disco rígido. Capacidade e velocidade.
Como conectar um disco rígido. Aplicações.

APLICAÇÕES

Planejamento de uma planilha eletrônica. Usos gerais.
Orçamento familiar. Digitação do programa.

PROGRAMAÇÃO DE JOGOS

As regras do *Jogo da Senha*. Definição das cores.
Escolha a melhor estratégia.

CURSO PRÁTICO **57** DE PROGRAMAÇÃO DE COMPUTADORES

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

